the TRS-80 MODEL 100 COMPUTER

The Micro Executive Workstation

KENNISTON W. LORD

THE TRS-80®
MODEL 100 COMPUTER:

The Micro Executive Workstation

KENNISTON W. LORD, Jr.

A Reston Computer Group Book
Reston Publishing Company, Inc.
A Prentice-Hall Company
Reston, Virginia

Library of Congress Cataloging in Publication Data
Lord, Kenniston W.
The TRS-80 model 100 computer.

"A Reston computer group book."
Includes index.
1. TRS-80 Model 100 (Computer) 2. TRS-80 Model 100 (Computer)—Programming. 3. Basic (Computer program language) I. Title. II. Title: T.R.S.-eighty model one hundred computer.
QA76.8.T184L67 1984 001.64'04 83-22804
ISBN 0-8359-7859-1

10 9 8 7 6 5 4 3 2 1

Printed in the United States of America.

Contents

Preface ix

Foreword xi

Trademarks xiii

PART 1

TOOLS FOR THE EXECUTIVE 1

CHAPTER 1

Features and Definitions 3

The Machine, 3
Vital Statistics, 4
Keyboard, 4
Function Keys, 6
The Screen, 7
Peripherals, 7
The Chip, 8
Intrinsic Functions, 9
Power . . ., 10
In Fact, 10

CHAPTER 2

Text Processor 11

Back to Text, 17
The Command Keys, 18
The Cursor Keys, 19
Building a Text File, 20
Editing Your Text, 21
Deletion, Duplication, and Movement of Text, 27
Using Tape Cassette Media, 29
Printing a Text Document, 30
Using CTRL Functions, 32
The GRaPH Key, 33

CHAPTER 3

Nonprogramming Capabilities 35

The Schedule Organizer (SCHEDL), 36
The Address Organizer (ADDRSS), 39
Telecommunications, 42

CHAPTER 4

On-Line Information Services 59

The Big Three, 59
All the Others, 62
CompuServe and Dow Jones Information Services, 69
USPS Electronic Computer Originated Mail (E-COM), 80

PART 2

TOOLS FOR THE PROGRAMMER 85

CHAPTER 5

Aw, Programming's Not So Tough! 87

Numeric Variables, 89
Stevens Fund Accounting, 92
Saving Your Program, 96

REMarkable Program, 100
STOP, END, and CONTinue, 101
Making the Process Automatic, 101
Determining the Size of Memory, 102
Error Conditions, 102
Math Operators, 102
Relational Operators, 103
The Editor, 107
Renumber, 108
If You Have Trouble, 108
Loop Control, 108
PRINTing to the Display Screen, 112
Tabulation, 116
Blinking Display, 117
The Printer, 117
CHR$, 118
The Print Mask, 120
GOTO, 121
INKEY$, 123
Summary, 124

CHAPTER 6

Developing and Manipulating Data **129**

Rules for Variable Names, 131
A Place for Everything, 132
Concatenation, 134
A DATA Statement, 134
Sentinel Line, 137
ARRAY, 138
DIM, 139
Strings and Things, 142
Binary Search, 144
Searching the Table, 149
Table Maintenance, 154
The Musical MEW, 155
Musical Inventory, 157
No Inclination—No TIME$, 172
Errors, 174
Communications Interruptions, 177
The KEYs to the Program, 178
Scroll, 180
Program Development, 181

RAM, 181
The POWER Function, 182
Data Conversion, 182
Negative Numbers, 183
Machine Address, 183
Other Kinds of Input, 183
Other Kinds of Output, 184
Whatever Makes Your MOTOR Run, 186
Summary, 186
And a Little Bit of Trig, 190

PART 3

MATH AND GRAPHICS **191**

CHAPTER 7

Screen Play **193**

Mathematics, 193
Trigonometry Begins with a Radian, 194
The Infernal Triangle, 194
COSine at the Bat, 195
Sound of the ArcTANgent, 196
It's No SINe, 196
And the Last Two, 197
Derived Functions, 197
Calculating Compound Interest—Math of Finance, 201

CHAPTER 8

Graphics **203**

Graphics in 3-D, 209
Line Graphs, 213
Block Graphs, 216
Summary, 218
One Final Word, 219

APPENDIX A

Instruction Repertoire **221**

APPENDIX B

Using Non-Radio Shack Printers 243

APPENDIX C

Computer-to-Computer Communications 245

Index 249

Preface

You've Come A Long Way . . .

Ten years ago, to make effective use of the travel time between New York and Sydney, Australia, I attempted to use a portable mechanical typewriter on the airplane. The experience proved to be intolerable to both the other passengers and myself, and while there were no power problems while down under, the size, weight, and inconvenience of the portable would preclude repeating the exercise. Even then the concept of a computer that weighed four pounds and fit easily into the briefcase was remote. Up to then—and indeed for yet another five years—the computer was a machine whose size could best be described as "larger than a breadbox," which required specialized power and air conditioning, and existed to serve many, not simply a writer occupying an airplane seat over the vast expanse of the Pacific Ocean.

The microcomputer revolution changed all that. First came the "desktop" microcomputer—an appliance, a business machine, fixed in nature. Then the microcomputer became portable; Radio Shack calls it simply "transportable." And then it became an accessory, as indispensable to many as the briefcase itself. Radio Shack has named the TRS-80 Model 100 a *Micro Executive Workstation*. That's good enough for abbreviation purposes, and we'll use the acronym MEW throughout the book.

However, let's not lose sight of the "accessory" nature of this device. This machine has more intrinsic (built-in) functions than *any* microcomputer in existence, as of this writing. That may change, as Radio

Shack is definitely breaking new ground. But we're talking about a computer whose power exceeds many of its larger brothers in a size not significantly different from a medium-sized notebook. In fact, I call this a "notebook" computer.

Thirty years ago the computer was larger than most houses and consumed enough power to light your neighborhood. It required enough air conditioning in a single day to cool your house for a week. Those who worked with the machine were trained in mathematics. The programs were literally wired into the machine. Today the computer weighs less than a sack of sugar, is programmed in simple nonmathematical language, and can communicate with other computers a room . . . a city . . . or a world away.

Computers, you've come a long way. . . .

In this volume the TRS-80 Model 100—the MEW—will come alive. Some of the information in this book is also available in the manuals. Some of it is brand new. We'll assume that the reader is a first-time computer user who has acquired the MEW as a business accessory. It's an accessory—a convenience that can enhance the way you conduct your business. It's a novel tool, a device to free up some of that valuable resource—time. And while it will make your available time more productive, it may well place demands upon that time.

This book will explore the device, its intrinsic software, and ways to use it that may not be immediately obvious. When you have mastered those things in this book, your experience with the microcomputer will be just beginning. If this book does what it should, the microbug will have bitten, and you'll be looking for other ways to scratch the itch.

For though it has come a long, long way . . . it still has a long way to go.

Kenniston W. Lord, Jr.

Foreword

This book is organized into three parts, each with a specific purpose.

In general, Part I contains discussion and use of all of the TRS-80 Model 100's intrinsic (built-in) functions except one: the BASIC Interpretive Compiler. Thus, in Part I you will find discussion about the TEXT processor (which will permit you to develop documents), the ADDReSS function (where not only names and addresses may be stored, but also telephone numbers which may be dialed automatically), the SCHEDuLe function (which is a handy reminder), and just enough BASIC to be useful.

The instruction manual that accompanies the TRS-80 Model 100 makes the assumption that the purchaser is knowledgeable in the BASIC computer programming language. In addition, it dodges responsibility by suggesting a most elementary book to be purchased if you don't happen to know BASIC.

Because there will be a software market for this machine, you'll have to learn how to use someone else's software; in addition, you'll no doubt want to write some software for yourself. Part II will teach you enough programming to get started. We won't make programming experts out of you—and we won't even discuss all of the available instructions. But we will, in this section, teach the rudiments of BASIC programming on this marvelous machine.

Part III deals with some advanced BASIC concepts—the mathematical functions and the graphics functions. While the machine is a

business machine, there are certainly some business people who need one or both of these capabilities. The treatment of these subjects will not be exhaustive, but will be a good starting point for your experimentation.

Trademarks

AUTO-CITE is a registered trademark of the Lawyers Co-operative Publishing Company and the Bancroft-Whitney Company.

Billboard Information Network is a registered trademark of Billboard Publications, Inc.

CitiShare Information Service is a registered trademark of CitiCorp.

CompuServe is a registered trademark of CompuServe, Inc.

DIALOG Information Retrieval Service is a registered trademark of DIALOG Information Services, Inc.

Dow Jones News/Retrieval is a registered trademark of Dow Jones & Company, Inc.

Electric Pencil is a trademark of Michael Shrayer.

EPSON is a registered trademark of the Epson Corporation.

KNOWLEDGE INDEX is a service mark of DIALOG Information Services, Inc.

LEXIS is a registered trademark of Mead Data Central.

MEDLARS is a registered trademark of the National Library of Medicine, National Institutes of Health, Department of Health and Human Services.

NewsFlash is a trademark of NewsNet, Inc.

NewsNet is a registered trademark of NewsNet, Inc.

NEXIS is a registered trademark of Mead Data Central.

PART 1

TOOLS FOR THE EXECUTIVE

CHAPTER 1

Features and Definitions

The Machine

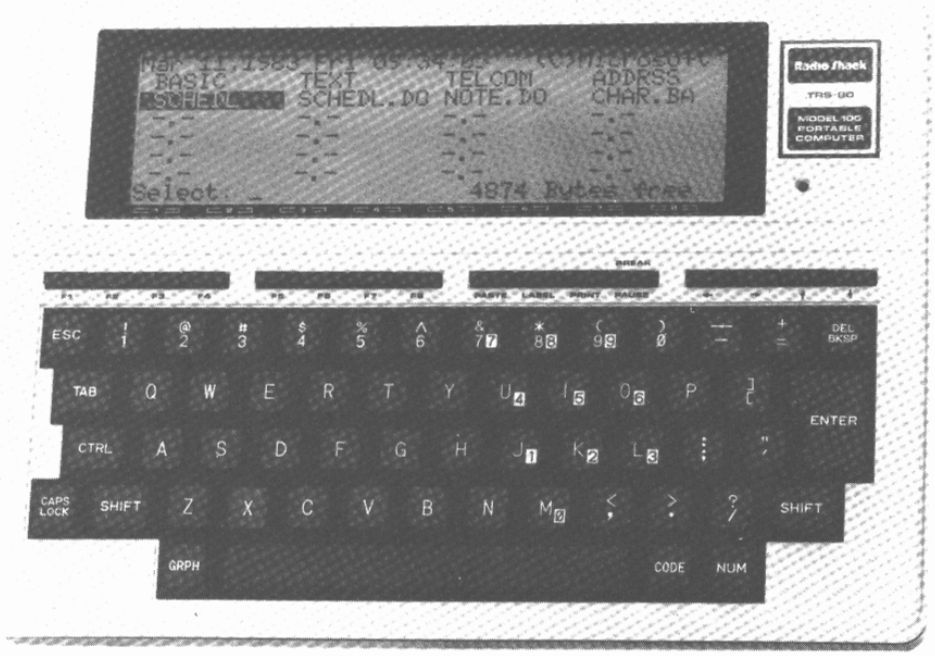


Figure 1-1. TRS-80 Model 100 Micro Executive Workstation.

Vital Statistics

The greatest news about the computer you have purchased—or whose purchase you contemplate—is its size. The overall size of the main unit (without peripherals) is the size of a small notebook: 2 × 11⅞ × 8½. That's about 164 cubic inches. An average loaf of bread, by comparison, would be more than 250 cubic inches. Its weight is less than four pounds—less than a small bag of sugar.

Keyboard

The next most striking feature is the size of the keyboard. This computer has a full-size typewriter keyboard. The keyboard is made up of depressionable keys, not membranes or buttons so small they must be pushed with a pencil. This computer can fit on the pull-down tray of an airline seat with room to spare. The keyboard on this computer will permit speed typing of text. And it is perfectly permissible to use this device on an airplane, by the way, as it is rated not to interfere with navigational instruments.

The standard typewriter keyboard is commonly called the QWERTY board, drawn from the leftmost five characters of the top letter row. There are other types of keyboards, but this is the most widely used. The keyboard has all the standard numbers and letters plus a few other features. On the left side of the keyboard, from top to bottom, are the following keys:

ESC — This is the ESCape key. When pressed, it generates a specific code known as ASCII 27. (We'll talk about the ASCII codes a little later.) The key is frequently used to abort the operation in which you are involved and return to a prior operation or to a place in the computer program where a choice of options (often called a *menu*) may be made. The key merely generates the code. It's the computer program's responsibility to recognize that code and to take specific action based upon it.

TAB — Like the TABulate key on the standard typewriter, this TAB key is used to skip horizontal spaces. Using this key you can indent sentence beginnings in a text file, subordinate instructions in a program, or whole subparagraphs in any document.

CTRL — This is the ConTRoL key. The CTRL key functions much like the ESCape key in that certain things oc-

cur when the key is depressed. Whereas the ESCape key requires that the program react when it has been depressed, the CTRL key causes the hardware to react. The CTRL key is used in conjunction with several other keys, each of which performs a specific hardware function when pressed in association with the CTRL key.

SHIFT — The SHIFT key functions just like the shift key on the typewriter. If you will look to the right of the keyboard you will find another SHIFT key. The purpose of the SHIFT key is to produce upper-case codes on the letter keys and special character codes on the number keys (top row) and on the punctuation keys at the right of the QWERTY board.

CAPS LOCK — The CAPS LOCK key is not a shift lock key. It causes a shift for the letter (alphabetic) keys only. There is no shift of the numeric or punctuation keys. You must still press the SHIFT key to achieve those. The purpose of the CAPS LOCK key is to provide total upper-case entry when desirable, for example, during the entry of a BASIC program.

GRPH — To the left of the space bar is the GRaPHics key. The MEW's display screen has 15,360 individually controllable picture elements (pixels) which are used for graphics. This key will be discussed later.

CODE — To the right of the space bar is the CODE key. As previously mentioned, the MEW uses the ASCII character set. ASCII stands for American Standard Code for Information Interchange, a code structuring system which was devised more than a decade ago and which is standard to all microcomputers everywhere. Again, that will be discussed later.

NUM — Note that ten of the keys on the QWERTY keyboard (M, J, K, L, U, I, O, 7, 8, and 9) are each marked with a number, 0–9 respectively. This is as close as the MEW will come to a numeric keypad. The numeric keypad is customarily used for the entry of numeric data and is easily used by anyone with adding machine experience. However, if these keys are merely depressed, letters will appear. The NUM key, to the right of the CODE key, is known as a *toggle* key, i.e., a switch which merely turns the keypad on and off.

DEL/BKSP — The DEL/BKSP key (Delete/Backspace) removes a character which has been entered by moving the *cursor* one space to the left, rubbing out the character which had been entered there. The *cursor* is a place marker on the screen for the next character to be entered. When used with the SHIFT key, the character located at the current position of the cursor is removed.

ENTER — This key may be thought of as a terminator. After you have typed a character, a line, a message, a paragraph, etc., you must signal the computer that nothing more will be entered at this time. You "terminate" that specific part of your conversation with the computer. Should you happen to draw information from other books or other computers, this key may often be referred to as CR, RETURN, <RET>, <CR>, or even LF.

Function Keys

Immediately above the QWERTY portion of the keyboard are 16 buttons, organized into groups of 4 buttons each. The leftmost 8 are function keys which may be used by some of the onboard (intrinsic) programs included with the MEW or with programs that you may develop. The remaining 8 keys perform specific functions either with the intrinsic programs or with the hardware itself.

The rightmost 4 buttons are cursor control keys. The buttons are marked with *left, right, up,* and *down,* respectively. Moving a cursor involves changing the focus of the computer to another specific place on the screen. Movement of the cursor does not destroy any characters located at those screen positions.

In the third group, known as command keys, one of the keys has two labels, BREAK and PAUSE. You will become quite familiar with this key since it interrupts the operation of a program or slows down the on-screen movement upward of information (called *scrolling*). The other command keys perform functions for the intrinsic software.

The PRINT key, you will discover, provides two printing abilities. Pressed by itself, the contents of the screen are committed to paper. Pressed in association with the SHIFT key, the entire contents of the document are committed to paper. One of the thrills of using this machine is to watch a full size printer being driven by this tiny device. We'll deal with the PASTE and PRINT keys, and the assigned function keys when we discuss the intrinsic software.

The Screen

The screen of the MEW is a Liquid Crystal Display (LCD). On the display are shown eight lines of 40 characters each. The screen functions as a window through which communication is passed. That window can be used to display memory directories, text which has been entered or stored, programs which are being developed or loaded, and messages from the hardware.

Like the LCD wristwatch, the characters are not easy to see, and the display angle has a lot to do with how well the information can be seen. Since it is not illuminated, as most displays are, the angle of the eye to the unit becomes important. Recognizing this, the manufacturer has provided a vernier adjustment which changes the viewing angle and intensity. Since it is not illuminated, however, the device is not something you'd work with in the dark. Characters are formed from a matrix of dots (5 × 7) at each of the screen positions. The height of the display is 64 dots; the width is 240 dots. Graphics are possible on this display, as well. There are 15,360 directly addressable picture elements (pixels). Text and graphics may be intermixed on the same screen. The size of the screen bears no relation to the size of any printed output, so using a screen line shorter than another system's screen line is no serious disadvantage.

Peripherals

It's got connections and then some! Here are some of the connections this machine has:

- *Parallel Printer Interface.* I stood in amazement as I watched this computer drive a full-sized printer. Any printer with the standard Centronics-type parallel printer interface may be used with this computer. (It drives the author's Epson MX-80.)
- *RS-232.* This provides capabilities to drive serial printers, plotters, communications devices, serial input devices, and a variety of interfaced peripheral devices.
- *Direct Connect Modem with Auto-Dialer.* We'll get to the auto-dialer later. The real news is that you can enter something into your computer and then transmit it to another device over the phone lines. This attachment permits direct connection to a modular phone jack, making your computer something like an extension telephone.
- *Acoustic Coupler.* This feature permits connection to another computer when direct connection is impossible.

- *Cassette Interface*. You may wish to temporarily store programs or text on some form of magnetic media. There is, after all, a physical limitation to what the available memory can hold. To accommodate that, a cassette interface has been installed to permit storage of that information on magnetic tape. It is entirely possible that by the time this book is published, some vendor will have devised a means to store data on diskette. These tapes can be read by other Radio Shack computers, incidentally.
- *Bar Code Reader*. This machine is certain to be used for bar code reading in inventory situations. Radio Shack now provides a reading wand.
- *Pin Connectors*. Future enhancements to this machine will interface with either the 28-pin connector for plug-in ROM—Read Only Memory—on which will be available other programming, or the 40-pin connector which will allow the MEW to interface with peripherals certain to come on the market, possibly even before this book is available.

The Chip

The key to the function of the TRS-80 Model 100, MEW, is its microprocessor, or chip. The MEW's microprocessor is known as a CMOS, or *Complementary Metal-Oxide Semiconductor*. Its actual name is 80C85, although it is referred to as 8085. The 8085 is a part of the series of processor chips which began with the 8080. The chip used in the IBM PC, for instance, is an 8088 and is a 16-bit chip. The 8085 is an 8-bit chip. The number of bits (contraction for *binary digit*) used by the microprocessor determines how much memory can be addressed.

The design point for the MEW is 32K of RAM—Random Acess Memory. K is a unit of memory containing 1,024 bytes (characters). Thus, the most memory that can be obtained with this system is 32 × 1,024, or 32,768 positions of memory. It may be obtained with as little as 8K, in a 24K version, and with an additional 8K add-on. RAM is the memory which will contain programs or documents you develop. Any contents of RAM memory will be lost when the power is interrupted. ROM, on the other hand, is *Read Only Memory*. The intrinsic functions are a part of ROM and the programs placed there are not volatile, i.e., they are not lost if power is interrupted.

Speed of the unit is something else—2.4 MHz. Speed really means little if everything is dependent upon your digital dexterity. But the difference lies with the speed of mathematical calculations. Speed of transmission over a communications line is a consideration, as well. Communications speeds of 300 to 19,200 baud are available. A baud may be thought of as *bits per second*. Communications, by the way, may be full

duplex or half duplex. A half duplex unit can both send and receive information, one way at a time. A full duplex unit can do both simultaneously. This computer can accommodate both.

Intrinsic Functions

It could easily be said that other computers provide capabilities, but the MEW provides the capabilities you need. The term has been used before, but it bears one more repetition—*intrinsic*. An intrinsic function is a function that is available within the unit itself. The MEW has these intrinsic functions:

- The word processing package won't challenge the prominent packages, such as Scripsit, Electric Pencil, WordStar, or VolksWriter, but it will provide a capable means to enter text (approximately 20 pages) for later transmission or transfer to a larger system or production to hardcopy printout. Lest that seem like a small number of pages, it should be remembered that we are dealing with *full* pages of text. The author of this book counts it a good day's work if 20 pages of text are produced. The capability, then, is not insignificant. MEW's word processor is called by a very original name: TEXT.
- Since this *is* a Micro Executive Workstation, those who use this device may well be executives whose work lives function according to a schedule. One of the intrinsic programs on the MEW is called SCHEDL. This scheduling program is a small file on which you will record appointments, keep track of on-the-road expenses, and keep your "to-do" list. The schedule may be reviewed on the screen or searched for.
- The person who needs to use an address book or other means to record addresses, phone numbers, names, etc., will find the MEW to be quite handy. As with the SCHEDL function, the ADDRSS function can be reviewed or searched for.
- Whenever it is necessary to communicate with another computer, the onboard TELCOM program comes into play. Using one of the communications services outlined here, communication may be made from your MEW to the home computer and vice-versa. It provides the basis for electronic mail, order entry, message file, report distribution, and a variety of other services. Networks now exist for such movement of data; more are certain to come into existence by the time this book is published. In addition you can tie into national information networks, such as the Dow Jones network, CompuServe, the Source. In fact, Radio Shack will give you the opportunity to try a couple of these free of charge.

- If you'd really like to do it yourself, the MEW offers a full-function Microsoft BASIC interpretive compiler as one of its intrinsic functions. You can write your own programs to perform functions not included in the machine. That will help you when you see something in a magazine or book which looks useful.

Power . . .

In and of itself, this computer is very powerful. But the package provides some other interesting features, as well. As odd as it sounds, it is possible to turn off this computer without turning it off. Generally, computer memory is volatile, i.e., whatever was in memory and showing on the screen disappears when the power has been removed. That is also true with the MEW, with one significant difference. There are two power switches: one turns off the display and one turns off the RAM.

RAM is kept active with nickel cadmium batteries which may be recharged by house current or by four AA size alkaline batteries. So long as RAM is energized, any information contained therein is safe. At home or in the office, the unit may be plugged into the wall. On the road, the alkaline batteries keep the Ni-Cads energized. A panel light will indicate when the batteries are low. Literature with the machine rates the refresh capability of the alkaline batteries at about 20 hours and the Ni-Cads at about 30 days. Replacement of the Ni-Cads will depend upon usage, approximately 30 months.

The on/off switch on the side of the unit controls the computer and the screen. A central power switch is at the bottom of the unit. You won't turn it off accidentally, as it's well protected.

In Fact . . .

This beauty has just about everything. And those are just the positive things. The negative things you'll have to deal with: the urge to work every available moment; the secret things which might accidentally get out of your computer; instantaneous access to information you really didn't want to know until next Thursday.

Ah, yes, the computer has certainly changed our lives. No more shorthand for secretaries to take. There's no need for it. The executive will probably draft the first copy anyway. No more sloppy notes for students to decipher. Those notes can be taken effortlessly and quietly at the back of the classroom. And you travelers to Sydney who have the urge to write—write on.

CHAPTER 2

Text Processor

We could label the TEXT feature of the TRS-80 Model 100 as a pint-sized word processor. To some extent, it has many of the text preparation features of a word processor. And it lacks certain features, such as line justification, indented margins, etc. However, if we can but recall that everything that goes into this machine is text, in one form or another, then it can be seen that this text processing module might well be one of the most powerful tools in this little package. In fact, as we shall see, TEXT is the default module. In the TEXT mode a document (identified by the suffix .DO) may be called for or created. However, once created, one need only pass the cursor onto that directory entry, press the ENTER key, and you are immediately in TEXT mode, within the document you have selected.

When you switch on your MEW, Figure 2-1 shows what you'll see.

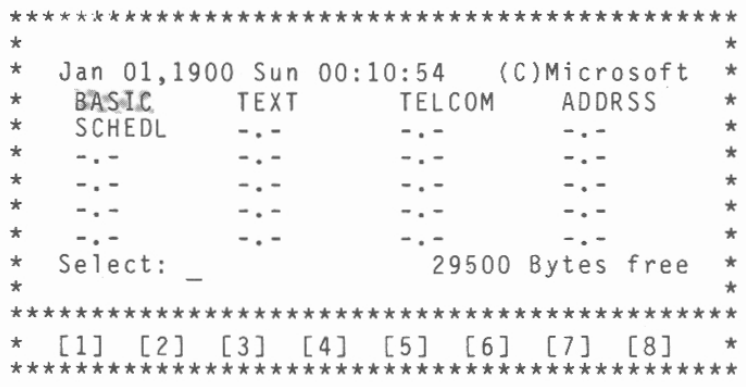


Figure 2-1

This is called the *directory,* or *MENU*. No matter what you are doing, the pressing of Function Key 8 (F8) will return you to this point. In the directory in Figure 2-1, note that the word BASIC has been shaded. Actually, this is called *reverse video,* which marks the location of the cursor. This says that the BASIC module is the one which is primed for selection. To select this module you would need only to press the ENTER key. Don't do it—just yet.

At the right, immediately above the keyboard, the rightmost set of buttons are the cursor movement keys. They indicate left, right, up, and down. Press them and watch what happens to the cursor. Note that if you continue to press either the right arrow key or the left arrow key, the cursor will move around the directory in rotation. However, this is not true of the up and down arrows. They will only go where there are entries. These entries are called *files*. Technically, the intrinsic programs are files, but later, when we get into BASIC and you seek to determine what files are in the machine, you'll note that the intrinsic programs will not be displayed.

One thing you'll discover in this package is the high degree of integration. The concept behind the design of the package is that if there is one way to do it, don't install a second method. To illustrate, note the first line of the directory:

```
*****************************************
*                                       *
*  Jan 01,1900 Sun 00:10:54   (C)Microsoft  *
```

This display shows the date and day when the unit was shipped. The time will reflect when you see it relative to when you turned on the machine. This particular display was captured 10 minutes, 54 seconds into working with the machine and before the information was reset. We will shortly change that.

Important point: There are two "on" switches for your MEW. The master switch, located on the bottom of the unit (recessed), energizes the Ni-Cads from either the batteries or from the AC power connection. How long the Ni-Cads hold their charge is a function of the installed memory. If you have purchased the 8K unit, you can expect the charge to remain about 30 days. If you have the 32K unit, you can expect the charge to remain 8 days. The switch at the right side of the unit merely turns the display on and off. Thus, if you enter a document to memory and turn off the switch at the right side of the unit, you will not lose your document from memory, so long as the Ni-Cads are charged. However, if you turn off the master switch, everything in memory will be gone except for the intrinsic programs.

Because of the integration previously mentioned, the date, day, and time are all set from BASIC; therefore, you must ensure that your cursor is set on BASIC and then press ENTER. This display will come up:

```
*********************************************
*  TRS-80 Model 100 Software                *
*  Copr. 1983 Microsoft                     *
*  20083 Bytes free                         *
*  Ok                                       *
*  +                                        *
```

The number of free bytes is a function of the memory purchased. This number is from the 24K version of the system. The cursor shown is actually a flashing block of reverse video, located in the same position.

Now find the command key marked *LABEL* and press it. Observe the bottom line of the screen:

```
*                                           *
* File Load Save Run  List           Menu  *
*********************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]   *
*********************************************
```

Note that if you press the LABEL key again, the line disappears. The LABEL key is said to *toggle*.

The words on the line correspond to the numbers in the line immediately beneath. These numbers are painted onto the black frame of the display. These numbers, in turn, correspond to the eight function keys, marked F1–F8 in the left two groups of function keys. F1 corresponds to *Files*. Press F1. If you have entered any documents, then they will show on the display. Thus the available files are displayed without leaving BASIC.

You will learn that *Save* (F3) stores a document on cassette tape. (Later enhancements may include disk.) *Load* (F2) brings the document in from the cassette tape. *List* (F5) will list a BASIC program. *Run* (F4) will cause a BASIC program to operate. Remember that F8 is the standard return to the MENU. Any of these can be typed at the BASIC prompt (Ok) at the location of the blinking cursor.

Don't push any of the function keys at this time, as we will set the calendar and clock while in BASIC. We will use these three commands, followed by ENTER:

```
DATE$ = "month/day/year"
DAY$ = "day"
TIME$ = "hour:minute:second"
```

These are known as BASIC *String Variables;* these three happen to be *reserved* string variables. A string variable is customarily used to hold nonnumeric information (as denoted by the quotes) and will always be named with a trailing dollar sign ($).

Thus, if you wanted the date to reflect Christmas Day, 1983, you would enter:

```
DATE$ = "12/25/83"
```

Since Christmas Day of 1983 falls on a Sunday, you would set the day of the week as:

```
DAY$ = "Sun"
```

using the following three letters: Sun, Mon, Tue, Wed, Thu, Fri, Sat.

Time is set on the 24-hour, or military clock. Thus the hours from noon are added to 12, e.g., 1 p.m. becomes 13; 5 p.m. becomes 17. Generally it is not necessary to set the seconds, but the hour and minute should be entered as follows:

```
TIME$ = "10:15:00"
```

In all these cases you have entered data in BASIC's *immediate* mode; that is, you have executed instructions directly. It just so happens you have entered a use of the LET instruction, which we'll discuss during the BASIC programming language sections of this book.

Hereafter, so long as you keep the batteries charged, the date and time will be as accurate when you turn on the screen as they were when you set them. Watch the clock for a while; note that the seconds flash as they pass by. So long as you have your MEW, you have a clock. We'll come back to it when we discuss BASIC, but here is a way to use it as your alarm clock:

```
10 CLS
20 PRINT TIME$
30 IF TIME$ = "07:00:00" THEN 50
40 GOTO 10
```

```
50 FOR N = 1 TO 500
60     PRINT CHR$(7)
70 NEXT N
80 END
```

If you're a light sleeper, this will wake you up. We'll explain this later. If you're doing nothing with the computer, then it will shut itself off in 10 minutes. If you're running this program, it will keep time and then begin "ringing" at 7 a.m. You should use the AC adapter when you do this.

We've done what we must in BASIC, now press F8 and move the cursor to TEXT. The screen will look like Figure 2-2.

```
*********************************************
*                                           *
*  Dec 25,1983 Sun 10:15:00   (C)Microsoft  *
*   BASIC      TEXT       TELCOM     ADDRSS  *
*   SCHEDL     -.-        -.-        -.-     *
*   -.-        -.-        -.-        -.-     *
*   -.-        -.-        -.-        -.-     *
*   -.-        -.-        -.-        -.-     *
*   -.-        -.-        -.-        -.-     *
*  Select: _                29500 Bytes free *
*                                           *
*********************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]   *
*********************************************
```

Figure 2-2

As you can see, we've assumed that you have received this book along with your TRS-80 Model 100 for Christmas, 1983, and you are sitting in your pajamas pushing buttons. Well, since the cursor is on TEXT, push the ENTER button. Immediately you'll see:

```
*********************************************
*                                           *
*  File to edit? _                          *
*                                           *
```

Enter your name (up to six characters) and press the ENTER key. The cursor should be at the upper left hand corner. Type to your heart's content. Fill up the screen and add at least one more line. Note how the screen jumps up. This is called *scrolling*. When you have finished typing, press F8 again. This will CLOSE your file and you will see your

name as the file name (up to 8 characters) with a suffix of .DO, for DOcument. The screen will now look like Figure 2-3.

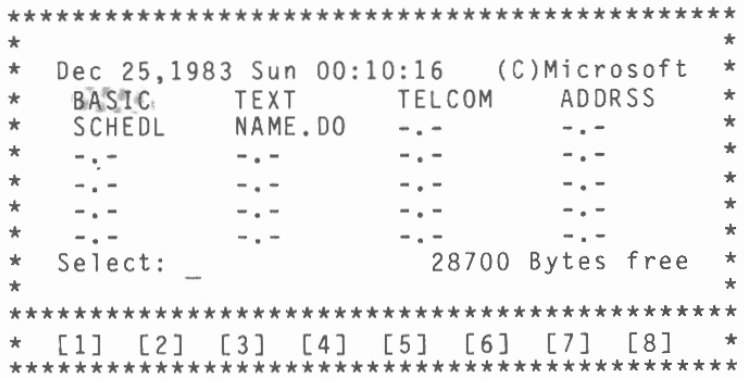


Figure 2-3

The number of bytes free will be a function of the number used in the document bearing your name, as well as the available memory in the computer. Now move the cursor down over the name of the document you entered, as shown in Figure 2-4.

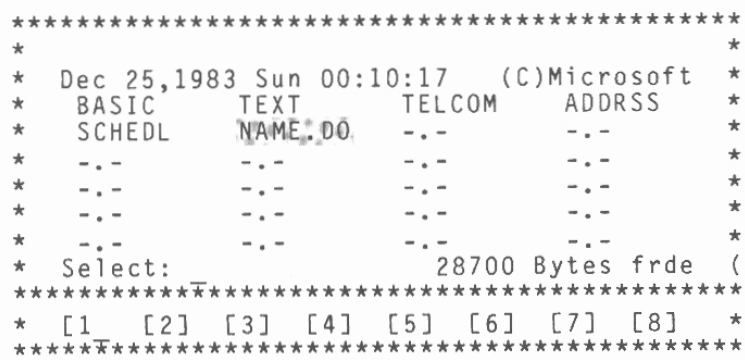


Figure 2-4

One more BASIC function must be learned before we can concentrate exclusively upon TEXT. Place the cursor onto the BASIC entry in the directory and press ENTER. When you receive the BASIC prompt and the block cursor, type this:

KILL "NAME.DO" <ENTER>

where name is the name you used to create the document in the first place. This is the only way—other than cutting the master switch on the back of the computer—for you to delete a file. Note that the name of the file is enclosed in quotation marks.

Back to TEXT

The function keys in the TEXT program are used the same as they were used in the BASIC program. You can see the functions assigned to the keys by pressing the LABEL command key once you have entered the TEXT program. These are the keys, followed by a brief explanation of each:

```
*                                              *
* Find Load Save      Copy Cut  Sel  Menu     *
************************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]     *
************************************************
```

F1 One of the nice features of the MEW is that you can locate a specific combination of characters anywhere in your document. Thus, if you are writing a letter and you want to locate the word "Dear," for example, the Find function will ask for the word and will then locate the word as many times as it occurs. The word can be spelled as shown, as "dear," or even as "deAr" and still be located. This feature is useful not only to locate a word in TEXT mode, but also in the ADDRSS and SCHEDL programs.

F2 The Load and Save functions of F2 and F3 work together. As the machine is currently configured, data, text, programs, or other documents may be stored on a cassette tape using a cassette tape recorder/player; this is done in audio tones. The Load (F2) reads the document from the cassette tape into the MEW's memory.

F3 The Save function places the information onto cassette tape in the first place. The machine contains the same machine language commands used by the TRS-80 Color Computer to store information on diskette. By the time this book is published, it is entirely possible that Radio Shack or someone else will have interfaced diskette drives to the Model 100.

F7 — Let's skip by the intervening function keys for the moment. For text to be moved, copied, or deleted, it must first be marked. The F7 key marks the text. More detail will be shown about how to use it when editing is discussed.

F5 — Once text has been marked, Copy will permit the duplication of that text at the location indicated by the cursor. It works in conjunction with the PASTE command key. In this instance, the text is simply copied to appear in both places.

F6 — Cut is used when the copy is to be removed from its originating location and placed in the target location. This, too, uses the PASTE function key. One helpful aspect is the ability, in TEXT, to copy or move text from one file to another.

F7 — F7 marks the text. F5 and F6 will load the buffer (F6 destroys the source). And PASTE places the contents of the buffer in the desired location in the same or another document.

F8 — Menu takes you back to the opening menu, closing your files in the process.

The Command Keys

PASTE — The PASTE key allows the insertion of text which has previously been defined, and COPYed or CUT. PASTE requires the cursor to be located at the target location.

LABEL — This key displays the current definitions of the function keys.

PRINT — This will cause the current contents of the display to be printed on the printer. If used in conjunction with the SHIFT key, the entire contents of the document are printed. Caution: the text produced by TEXT *does not* include a line feed at the end. Those printers which do not require a line feed (this unit was tested on a Radio Shack DWP-410 and a Radio Shack Line Printer IV) may be used. Use of other printers requires you to preset the machine. Tests on an Epson

MX-80 showed that the machine would drive the printer acceptably—as long as everything is printed on the same line. Printing on that printer could be accomplished by internal switch changes, however. Because text is not formatted on the screen of this text processor, you must specify a WIDTH for your printed output. If your printer is an 80 column printer, then respond 80 and press ENTER. If the width is different, respond with the appropriate number. You are not limited to the physical width of the printer, however. If you want to produce the document in 20 column widths, simply enter the number 20. A BASIC program to print TEXT documents using other printers is at the end of the book.

BREAK — The command key marked BREAK/PAUSE is used to arrest the forward progress of a display. When used in conjunction with the SHIFT key, it will interrupt the process. First press the BREAK/PAUSE key. Next, while pressing the SHIFT KEY, press the BREAK/PAUSE key again.

The Cursor Keys

You already know what happens when you press the cursor keys: the cursor moves one character to the right or left or one line up or down. The cursor keys work also with the SHIFT and CTRL keys, when that key is pressed before the cursor key, with the following results:

SHIFT:
- Right, to the next work.
- Left, to the previous word.
- Up, to the top of the display in the same relative location on the line.
- Down, to the bottom of the display in the same relative location on the line.

CTRL:
- Right, to the end of the current line.
- Left, to the beginning of the current line.
- Up, to the beginning of the *file*.
- Down, to the end of the *file*.

Building a Text File

Naturally, you can't edit, search, print, or find anything in a text file which doesn't exist. So, from the main menu, place the cursor over TEXT and press the ENTER key. This is what you'll see:

```
*********************************************
*                                           *
*  File to edit?                            *
*                                           *
```

That looks familiar. Give it a name—like CHAP1—and press the ENTER key. When that has been done the display screen will clear and you can begin entering information at the cursor position in the upper left hand corner. Any length line can be entered; press ENTER at the completion of your entry. However, if the sentence you are entering is longer than 40 characters, it will break at a word ending and resume at the beginning of the next line. Because of a feature called *wordwrap,* it is not necessary to press ENTER at the end of an incomplete line.

So that you can see what a screen looks like and use some of the editing features of the MEW, enter the quotation shown in Figure 2-5 from another of the author's books, *Using the IBM PC,* published by Van Nostrand Reinhold:

```
*********************************************
*  Forty years ago there was no such thing  *
*  as an electronic computer, as we know    *
*  them today.  Forty years ago we were     *
*  embroiled in World War II and            *
*  electronics development was just coming  *
*  of age.                                  *
*                                           *
*                                           *
*********************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]   *
*********************************************
```

Figure 2-5

Since this is a short paragraph, it is not necessary to hit the ENTER key until the very end. Had you entered more than eight lines, each line would have shifted upwards. It would have been necessary to move around in the document using the cursor keys by themselves or in conjunction with SHIFT or CTRL keys, as previously discussed.

When you have completed your entry, close your file by pressing F8, the Menu option. That will save the document in memory and return you to the main menu. You can exit to the main menu via the ESCape key, but the file will not be saved to memory. If you close the TEXT processor via F8, your screen will look like Figure 2-6.

```
*********************************************
*                                           *
*  Dec 25,1983 Sun 11:00:00   (C)Microsoft  *
*   BASIC      TEXT       TELCOM     ADDRSS   *
*   SCHEDL     CHAP1.DO   -.-        -.-      *
*   -.-        -.-        -.-        -.-      *
*   -.-        -.-        -.-        -.-      *
*   -.-        -.-        -.-        -.-      *
*   -.-        -.-        -.-        -.-      *
*  Select: _                xxxxx Bytes free  *
*                                           *
*********************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]   *
*********************************************
```

Figure 2-6

The bytes free will be determined from the available memory and whatever else you have stored in memory. Note that we've been generous and given you an additional 43 minutes to open your other presents.

Editing Your Text

Adding information to your text is extremely easy. Just position the cursor where the insertion is to be made and begin typing. But you had closed your document. Move your cursor over the document name and press ENTER. The document, as you had typed it, will now appear. We're going to add a sentence to the end: "The first commercial computer would not hit the market until 1951."

So, press the CTRL key and the down arrow, and the cursor will be moved to the end of the file you had previously entered. At this point, simply add the new sentence as shown in Figure 2-7.

A significant feature of the editing capabilities of the Model 100 is its ability to mark text for copying, deletion, or movement. This, you will recall, is done through a *buffer,* or a temporary storage location. In

```
*******************************************
*  Forty years ago there was no such thing  *
*  as an electronic computer, as we know    *
*  them today.  Forty years ago we were     *
*  embroiled in World War II and            *
*  electronics development was just coming  *
*  of age.  The first commercial computer   *
*  would not hit the market until 1951.     *
*                                           *
*******************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]   *
*******************************************
```

Figure 2-7

the MEW it is called a PASTE buffer. The first step in marking the text is to position the cursor on the first letter of the first word to be marked. For illustration, let's work with the *middle* sentence in the sample illustration, CHAP1.DO. Therefore, using the cursor movement keys, position the cursor at the "F" of the word "Forty." Now press F7, Select. You have told the computer that from that point you will mark text. Now move the cursor and you will see that the text is marked in reverse video. If you want to delete it, press F6 (Cut) and it will disappear. If you want to copy it, press F5 (Copy), move the cursor to where you would like the copy to be, and press PASTE. If you wish to reverse the definition process, to "undefine" the text, simply press BREAK followed by the SHIFT and BREAK keys simultaneously. As mentioned previously, you may even go back to the main menu and add what is now in the PASTE buffer to another document. Try this last example, but leave the original document undisturbed.

When marking text, you can move the cursor from character to character, or you may use the cursor keys in conjunction with the SHIFT and CTRL keys. Each of the following assumes a fresh load of your document, that you have positioned your cursor under the "F" of "Forty" in the second sentence, and have pressed F7 (Select). From that point you can cut, copy, or paste as needed.

Mark by word: SHIFT and right cursor, to the next word. This would mark the word "Forty," which begins the second sentence (see Figure 2-8).

Here only one word is marked. However, it is desirable to mark the entire sentence. Note that his sentence takes a part or all of four individual lines. Therefore some other action will be required.

Mark to the end of the line: Press F7 to begin the marking where you are and then, in combination, press the CTRL and the right

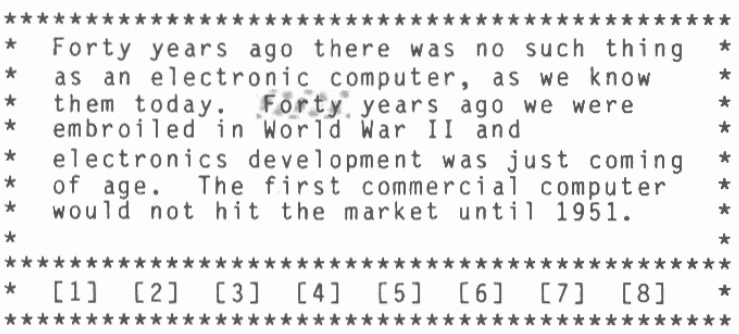


Figure 2-8

cursor. Note that the reverse video now extends to the end of the line. Now, press the "down" cursor and you will see that the reverse video has been extended to the next line. Do it again and the third line will be reversed. Since we're interested only in two words on the third line of the second sentence, move the cursor to the beginning of that line. Then in conjunction with the SHIFT key press the right cursor twice. Once F7 has been pressed, it need not be pressed again until you have completed the action with a Cut or Copy function.

Using the CTRL key or the SHIFT key in association with cursor keys, large blocks of text can be marked. Bring in the document again and position the cursor at the same location. Try each of these to become familiar with marking:

- SHIFT, left cursor, which marks one word to the left (see Figure 2-9).

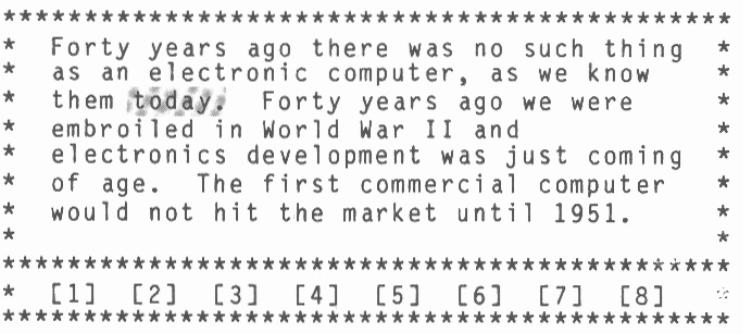


Figure 2-9

- SHIFT, right cursor, which marks one word to the right (see Figure 2-10).

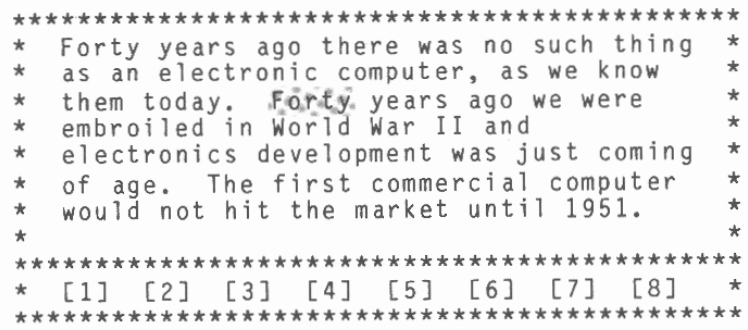


Figure 2-10

- SHIFT, up cursor, which marks to top row, same relative position (see Figure 2-11).

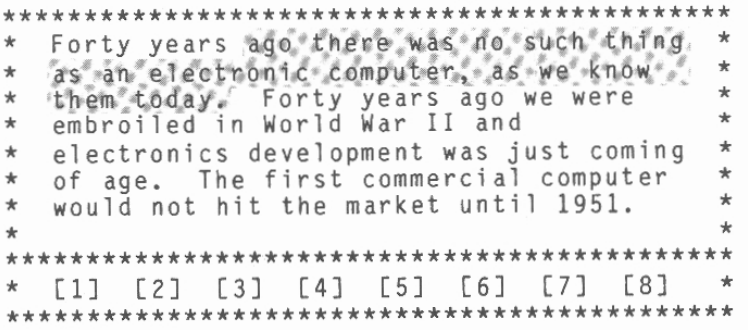


Figure 2-11

- SHIFT, down cursor, which marks to bottom row, same relative position (see Figure 2-12).

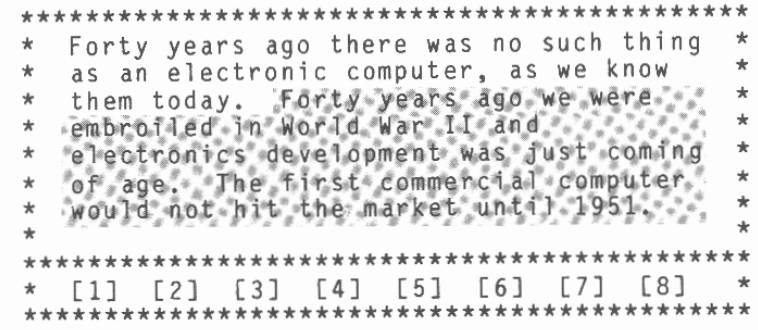


Figure 2-12

- CTRL, left cursor, which marks to the beginning of the line (see Figure 2-13).

```
*********************************************
*  Forty years ago there was no such thing  *
*  as an electronic computer, as we know    *
*  them today.  Forty years ago we were     *
*  embroiled in World War II and            *
*  electronics development was just coming  *
*  of age.  The first commercial computer   *
*  would not hit the market until 1951.     *
*                                           *
*********************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]   *
*********************************************
```

Figure 2-13

- CTRL, right cursor, which marks to the end of the line (see Figure 2-14).

```
*********************************************
*  Forty years ago there was no such thing  *
*  as an electronic computer, as we know    *
*  them today.  Forty years ago we were     *
*  embroiled in World War II and            *
*  electronics development was just coming  *
*  of age.  The first commercial computer   *
*  would not hit the market until 1951.     *
*                                           *
*********************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]   *
*********************************************
```

Figure 2-14

- CTRL, up cursor, which marks to the beginning of the document (see Figure 2-15).

```
*********************************************
*  Forty years ago there was no such thing  *
*  as an electronic computer, as we know    *
*  them today.  Forty years ago we were     *
*  embroiled in World War II and            *
*  electronics development was just coming  *
*  of age.  The first commercial computer   *
*  would not hit the market until 1951.     *
*                                           *
*********************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]   *
*********************************************
```

Figure 2-15

- CTRL, down cursor, which marks to the end of the document (see Figure 2-16).

```
*********************************************
*  Forty years ago there was no such thing  *
*  as an electronic computer, as we know    *
*  them today.  Forty years ago we were     *
*  embroiled in World War II and            *
*  electronics development was just coming  *
*  of age.  The first commercial computer   *
*  would not hit the market until 1951.     *
*                                           *
*********************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]   *
*********************************************
```

Figure 2-16

Now that you know how to mark the text, we'll work with the other functions.

At this point, bring back the label display, which should look like this:

```
*                                           *
* Find Load Save      Copy Cut  Sel  Menu   *
*********************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]   *
*********************************************
```

While the methods described previously will certainly mark text, they are not the only way to do so—and, in fact, there may be an easier way to do so: using the Find function (F1). This function can be used to search for text unilaterally or, if used in conjunction with F7, can be used to mark text. Bring up the text example we have been working with. At the beginning of the text, the cursor is resting on the "F" of the first word of the first sentence. "Forty."

We will mark the final sentence of the text: "The first commercial computer would not hit the market until 1951." Look the text over and you will see that the word "The" does not occur anywhere else in the text, though there are several occurrences of "the." Now, press the Label command key so you can see the available functions and then press F1, "Find." Because the computer doesn't yet know what to find, it will respond with the message "String:", followed by the flashing cursor.

Now experiment. Type the string "the" as the characters you wish to locate and press the ENTER key. The cursor will first come to rest on the word "there," as the first three letters fit the requested string configuration. Press F1 again and you will see that the requested string

now exists immediately beside the question. Press the ENTER key again and you will come to rest on "them." Repeat this again, and you will find "The." And finally, you will find the word "the" in the last sentence. The important thing to remember is that the find string "the" would locate either "the" or "The." Now, repeat the process for a search string which looks like "The," and note that the same thing occurs. While this may seem like an inconvenience, the power of the system will be more than sufficient for most needs. The problem can be solved simply by extending the length of the search string to "The first," a combination found nowhere else in the text.

Move to the "The" of the final sentence by successive uses of F1. Now press F7 to mark the first word. Next, press F1 and enter "." as the search string. Once you have pressed the ENTER key, you will find text marking will be complete up to, but not including, the period. Move the cursor one position to the right and you will have everything. You may now use the marked text to either cut or copy, with or without the Paste option.

Deletion, Duplication, and Movement of Text

The deletion of text may occur with the DEL key in association with the SHIFT and BKSP key if you wish to do the deletion one character at a time. For larger blocks of information, however, the CUT option, F6, is used.

The purpose of the Cut option is to remove the text from its original place, after that text has been marked by methods just described. When you CUT the text, that text is placed in a PASTE buffer, a temporary storage location for the marked text. Should you choose not to PASTE that text back into the document somewhere else—or into another document—the net result is that the text is removed. Should you wish to cut and paste, however, the process requires that you (1) mark the text, (2) cut the text using F6, and (3) move the cursor to the destination location, and (4) paste the text using the PASTE command key. Figure 2-17 shows what the process will look like.

Now, position the cursor at the top left corner of the screen (CTRL and the "up" cursor) and press F7 to mark the beginning of the text. Next, find (F1) the period at the end of the sentence by specifying the space following the period (". ") as your search string. And, finally, press F6 to cut the text. You will find that the marked text is removed and the remaining text moves up to fill in the empty space. Figure 2-18 shows the before and after results.

It's important to realize that the marked text which has been CUT remains in the PASTE buffer for you to apply elsewhere. It will be in that buffer until you CUT the next marked text.

```
*********************************************
*  Forty years ago there was no such thing  *
*  as an electronic computer, as we know    *
*  them today.  Forty years ago we were     *      F6 (CUT)
*  embroiled in World War II and            *
*  electronics development was just coming  *
*  of age.  The first commercial computer   *
*  would not hit the market until 1951.     *
*                                           *
```

```
The first commercial computer would
not hit the market until 1951.
```

```
*********************************************
*  The first commercial computer would not  *
*  hit the market until 1951.  Forty years  *
*  ago there was no such thing as an        *
*  electronic computer, as we know them     *      PASTE
*  today.  Forty years ago we were          *
*  embroiled in World War II and            *
*  electronics development was just coming  *
*  of age.                                  *
*                                           *
```

Figure 2-17

```
*********************************************
*  The first commercial computer would not  *
*  hit the market until 1951.  Forty years  *
*  ago there was no such thing as an        *      CUT
*  electronic computer, as we know them     *
*  today.  Forty years ago we were          *
*  embroiled in World War II and            *
*  electronics development was just coming  *
*  of age.                                  *
*                                           *
```

```
*********************************************
*  Forty years ago there was no such thing  *
*  as an electronic computer, as we know    *
*  them today.  Forty years ago we were     *
*  embroiled in World War II and            *
*  electronics development was just coming  *
*  of age.                                  *
*                                           *
```

Figure 2-18

Using Tape Cassette Media

Text developed using this text processor may be saved on cassette tape and returned to the computer when needed. While a specific cassette tape recorder is recommended, nearly any good quality audio cassette tape recorder will work, provided you have obtained the proper connection cables and provided you follow a few simple steps.

Connecting the tape recorder involves using the cable which has a single connection on the computer side and a triple connection on the recorder side. Connections to the tape recorder are as follows:

Black plug—EAR connector

Large gray plug—AUX connector

Small gray plug—REM connector

One connection controls the motor, one is the data path inward, and one is the data path outward.

The cassette recorder functions independently of the computer insofar as its physical controls are concerned. To load text from the cassette recorder, press the PLAY button on the recorder. The cable connected to the EAR connector will "hear" the text data on behalf of the computer. To save text on the cassette tape, press both the RECORD and PLAY buttons, and the cable connected to the AUX connector will function as the computer's input to the recorder. The REM controls the turning on and off (at the computer's command) of the recorder's motor, *provided* the recorder's external controls have been properly set.

You must ensure that the tape is blank before you begin to record on it, that you have moved beyond any plastic leader before you use it, and that the controls are properly set.

```
*                                                    *
* Find Load Save      Copy Cut  Sel  Menu   *
******************************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]   *
******************************************************
```

To store text:

- Ensure that you have opened the proper text file by moving the cursor onto that directory entry and pressing the ENTER key.
- Ensure that the tape recorder is properly set up with the RECORD and PLAY buttons depressed.
- Press F3 (Save).

- When the prompt "Save to" appears you must provide a *unique* file name, not to exceed six characters, which will identify the file being saved. That name will be used to retrieve the file.
- Press ENTER. The command will start movement of the recorder and the transfer of data. That movement will cease when all the data have been transferred.

To load text:

- Set the volume control of the recorder to a place between setting 4 and 6. Some experimentation will show whether or not the setting is accurate. Dropout of data generally means it is set too low. Garbling of data generally means it is set too high. If your recorder has a tone control, or a Hi/Lo switch, set it to high or treble.
- Ensure that the tape recorder is properly set up with the PLAY button depressed.
- Press F2 (Load).
- When the prompt "Load" appears you must provide the *unique* file name, not to exceed six characters, which was used to identify the file when it was being saved. That name will be used to retrieve the file.
- Press ENTER. The command will start movement of the recorder and the transfer of data. That movement will cease when all the data have been transferred. Two other messages will appear as this operation is performed. If there are several files on the tape, the computer will indicate that it is skipping over other documents by the message "SKIP: filename," where the filename is the file name of other documents. When the requested document has been found, the message will be "FOUND: filename."

Printing a TEXT Document

One of the very nice features of the MEW is its ability to transfer the text you have entered to a printer. To do so you must have two things: a cable and a printer. Naturally, these are designed to interface with Radio Shack printers, though they can be interfaced to other similar printers. However, the Radio Shack printers provide automatic linefeeding at the end of each printed line. Other printers can only do so provided the electronics are properly switched. You would be wise to check the interface between the computer and the printer you wish to use should this present a problem.

The cable connects to the back of the computer in the slot marked "Printer." At this time, only Radio Shack provides the cable (Part Number 26-1409), which is not expensive. The standard cable fits what is known as the Centronics Parallel Port connection, and experimentation with this cable and the computer shows that it will drive a variety of parallel port-oriented printers. You will learn, with more experience, that the MEW can also drive serial printers through the RS-232 connector, located beside the parallel port. The RS-232 is the serial interface connection, useful for a variety of tasks.

Printing with the TEXT processor has two options:

1. Print only what is on the screen by pressing the PRINT command key.
2. Print the entire text file by pressing the SHIFT key and the PRINT command key simultaneously. When doing so, the processor will request a width, which is either the column width of your printer or the column width of the printed output, whichever is greater or desired. If your printer is 80 columns, answer 80; if it is 132, answer 132. That count will become the default for each time you use it until you change it. If it is your desire to print 20 character columns (e.g., for magazine paste-up), you have only to specify 20. It won't print your text side by side, but what do you want for a thousand dollars?

Each printer you will use will reserve certain codes or escape sequences to cause things to happen. These will allow underlining, bold face, enlarged print, diminished print, etc., assuming your printer has such capabilities. On the Radio Shack DWP-410 printer, for instance, the ASCII code to begin underlining is 15 and to cease underlining, it is 14. (ASCII is an acronym for the American Standard Code for Information Interchange, the coding system used in microcomputers.)

To achieve the value of 15 as an underlining command on the MEW, follow this sequence: CTRL P CTRL O. These will show on the screen as ^P^O. You must hold down the CTRL key and press P (for "printer") and O (for "underline") immediately preceding the place where the underline is to occur. To cease underlining, the sequence is CTRL P CTRL N (shown as ^P^N). While these codes will perform the functions indicated, they will not perform them in the PRINT (print only the screen) and SHIFT/PRINT (print the entire document) functions. In those functions these codes will be ignored. To achieve these functions, "save" the document to the printer rather than to cassette tape. When the request for document name comes up on the screen, you must respond with the reserved name LPT:, which stands for *line printer*. In this instance, the WIDTH option is not used. Thus you must do all the necessary formatting within the text.

Using CTRL Functions

Every function of the keyboard is duplicated via ConTRoL keys. As you grow more proficient with the MEW, you may find it more natural to use the control key functions. For the moment, experiment with the functions to see how they work. The following list details the control codes. All these codes assume that the CTRL key is pressed in conjunction with use of the codes:

A	Moves the cursor from its current location to the first letter of the first word to the left of the current location.
B	Moves the cursor from its current location to the bottom of the display.
C	Cancels a SELECT, SAVE, LOAD, FIND, or PRINT function. In fact, the ^C is widely used on most microcomputers as a BREAK function.
D	Moves the cursor from its current location to the position one character to the right.
E	Moves the cursor from its current location to the same location one line above.
F	Moves the cursor from its current location to the first character of the next word to the right.
G	Saves a file or a program.
H	Deletes the previous character.
I	TAB.
L	This is the same as the SELECT function key.
M	Carriage Return and Linefeed.
N	This is the same as the FIND function key (not to be confused with its use in association with CTRL P).
O	Same as the COPY function key (not to be confused with its use in association with CTRL P).
P	Sets up the printer, as previously discussed.
Q	Moves the cursor from its current position to the leftmost position on the current line.
R	Moves the cursor from its current position to the rightmost position on the current line.
S	Moves the cursor from its current position to the character next to the left.

T	Moves the cursor from its current position to the top of the display in the same position on the line.
U	Same as the CUT function key.
V	Same as the LOAD function key.
W	Moves the cursor from its current location to the first character of the file, moving the opening text into the display area.
X	Moves the cursor from its current location to the same position in the following line.
Y	Prints the entire file.
Z	Moves the cursor from its current location to the end of the file, moving the closing text into the display area.

If you will look into the manual's appendices, you will find that these are the first few characters in the ASCII character set. In fact, the first 32 characters of the character set are Control functions. In BASIC, these will perform other functions, as well.

The GRaPH Key

While looking at the ASCII character set, look at those beginning with decimal number 123. From 123 through 255 are graphics characters which may be achieved several ways. One way to achieve these is by pressing the GRPH key and the keyboard character indicated. While these will appear on the screen, and might also appear on a graphics printer, they will not show with the majority of letter-oriented matrix or daisy-wheel printers.

CHAPTER 3

Nonprogramming Capabilities

Radio Shack's description of the MEW as an executive's workstation is certainly appropriate, for it provides the executive with accessory tools which will both assist the executive's performance and provide relief from office panic situations wherein everything must be done now, or yesterday, if possible. The device is extremely useful for anyone who requires portability when capturing information—even to those who haven't discovered that such capability is useful or even necessary.

The previous chapter was devoted to the first of the MEW's five intrinsic software systems, the TEXT processor. By *intrinsic,* we mean that these are *burnt in* to Read Only Memory (ROM), such that they are always present when the device is turned on. This chapter will concentrate on the balance of the applications *other than* the BASIC programming language, which will require extensive discussion. Applications discussed in this chapter include:

SPECIAL ADAPTATIONS OF THE TEXT PROCESSOR

The personal schedule (SCHEDL) package is used to store your schedule, so you may sequence your appointments and search on the file. Again, this contains the ability to classify the file.

The address book (ADDRSS) package is used to store your name and address file, and telephone numbers. And, using TELCOM, the communications package discussed next, this machine can dial the stored telephone numbers. (You'll have to talk for yourself.) You have also the ability to classify your file.

COMMUNICATIONS CAPABILITIES

The telecommunications package (TELCOM) is the onboard communications processor which takes text, addresses, phone numbers, and other information and transmits it across communications lines or cables for handling by other computing hardware. This package works with the onboard MODEM, outboard direct connection cable, and the acoustic coupler. And it can handle incoming (uploading) or outgoing (downloading) information.

The first two are functions of the TEXT processor. They are not the same; certain requirements of the ADDReSS organizer and the SCHEDuLe organizer are fulfilled by the TEXT processor. Radio Shack has wisely integrated functions on these intrinsic packages, so as not to duplicate functions. Therefore you should know something about each, for they are somewhat interdependent, in the same way you should know something about the BASIC language built into the machine to delete a file which was developed by another application.

The telecommunications package is different in capability, although similar in function. It exists as the communications vehicle which ties together those documents created within the system or to be received by the system. It performs many unique functions, which will be discussed later.

The Schedule Organizer (SCHEDL)

The Schedule Organizer program (SCHEDL) is a special form of the TEXT application; with it you can store dates, times, and any necessary additional information. SCHEDL uses a special file, a unique file called NOTE. To use SCHEDL initially, you must build the document NOTE using the TEXT processor. From the main menu, enter TEXT, just as we discussed in the previous chapter. When TEXT asks you for a file name, respond with "NOTE." TEXT will change that to NOTE.DO. Once NOTE.DO has been created, it will be necessary only to move the cursor over NOTE.DO and press the ENTER key to add information, delete information, or search upon it. All the rules of TEXT will apply during this operation, including all cursor functions, function key functions, and command key functions.

SCHEDL, then, exists as a search operation upon the data entered.

The kinds of information kept in the SCHEDL file NOTE.DO can go beyond normal scheduling concerns. For example:

$ 09/01/83—Pay Club Dues—$50

@ 09/05/83—9:30 Meeting with Ben Brown

% 09/10/83—Quarterly Progress Report due

09/15/83—Sandi's Birthday

When you have finished entering your schedule and any classification information, press F8 (Menu) which will close the file and return a display which looks like Figure 3-1.

```
*********************************************
*                                           *
*  Jan 01,1900 Sun 00:10:54   (C)Microsoft  *
*   BASIC     TEXT      TELCOM    ADDRSS    *
*   SCHEDL    NOTE.DO   -.-       -.-       *
*   -.-       -.-       -.-       -.-       *
*   -.-       -.-       -.-       -.-       *
*   -.-       -.-       -.-       -.-       *
*   -.-       -.-       -.-       -.-       *
*  Select: _              29500 Bytes free  *
*                                           *
*********************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]   *
*********************************************
```

Figure 3-1

As was stated, SCHEDL is the searching mechanism upon the devoted file NOTE.DO. Move the cursor to SCHEDL and press ENTER.

Searching this file for anything containing the characters "Sandi" would produce a list of all occurrences of "Sandi" on the screen. If Sandi is a member of the office staff, however, and it were important to detail all data applicable to your office staff, then it would be necessary to search for only "#". Expense account data may be located by "$". Meeting information may be located by "@". Work items could be located by "%". The assignment of classification character is your decision. It would be wise to develop a set of meaningful classification rules. Locate the appointments or reminders for a specific date or group of dates simply by entering the date. Anything for the month of July could be located by finding "09/". This assumes that you have been consistent in the format of the data which has been entered, a *must* if searching for data is among your desires.

When you invoke the SCHEDL program, if the NOTE file has not been created, you will receive this message, accompanied by a beep from the computer's speaker:

```
NOTE.DO not found
Press space bar for MENU
```

If you receive the message, simply press the spacebar and you'll be back at the beginning. From that point it will be important to develop the NOTE.DO file using the TEXT processor. SCHEDL uses NOTE.DO, but doesn't create it.

When you enter SCHEDL, initially these three function keys will be assigned:

```
*                                              *
* Find                  Lfnd            Menu   *
************************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]      *
************************************************
```

The Function Key functions are:

F1 (Find) permits a text search to be performed. It works in the same manner as it does in the TEXT processor, whereby you identify a name, date, classification, or other pertinent data from the NOTE.DO file.

F5 (Lfnd) works just the same as F1, except that the located item is printed on a connected printer, rather than on the display. In this manner, when you indicate that no specific string is desired (press ENTER), your schedule file will simply print out on the printer. You will prefer to use the PRINT command key with the SHIFT key to perform this function.

F8 (Menu) returns you to the opening menu of the application.

The Command Key functions are:

PASTE Allows insertion of text which had previously been COPYed or CUT. Thus, if you have been developing a second schedule, perhaps for the next month under another name, such as SCHEDL.DO, you could simply cut that portion of the second schedule and move it into NOTE.DO in this manner. This feature should be used frequently, as the smaller the NOTE.DO file can be kept, the more efficiently the search mechanisms of SCHEDL will be, to say nothing of the amount of data that can be displayed.

LABEL	Displays the function keys used with SCHEDL. This key simply toggles the descriptive labels on and off. If toggled on, they occupy the bottom line of the screen. If toggled off, that line is available for file data.
PRINT	Prints the display screen on the printer. When used with SHIFT, the entire file may be printed. This is perhaps the more efficient way to list the file.
BREAK	Press BREAK (SHIFT/PAUSE) TO cancel operations.

The use of Find or Lfnd really produces only the next six occurrences of the search string. The operation must be repeated to locate the second set of six. Each occurrence may be the last, of course, but since there is no "find all" denotation of this function, it must be repeated. The MEW assists this requirement by displaying a second set of prompts:

```
*                                                *
*            More Quit                           *
**************************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]   *
**************************************************
```

Since more information may be displayed on the screen, F3 (More) will permit the next screen (6 lines) to be displayed or printed, depending upon whether you initially selected Find or Lfnd. F4 (Quit) will return you to the initial display of function keys where the process may be continued, or you may return to the main menu. That's all there is. It is no more complicated than that.

The Address Organizer (ADDRSS)

The Address Organizer program (ADDRSS) is another special form of the TEXT application. The idea is to store names, addresses, telephone numbers, and sufficient additional information which may be needed in a uniquely named file, called ADRS.

Like NOTE, ADRS must be built using the TEXT processor. Therefore, from the main menu, it is necessary to enter TEXT. When TEXT asks for a file name, respond with "ADRS." TEXT will change that to ADRS.DO. Once ADRS.DO has been created, it will be necessary only to move the cursor over ADRS.DO and press the ENTER key to add information, delete information, or search upon it. All the rules of TEXT will apply during this operation. ADDRSS, then, exists as a

search operation upon the data entered. Also, this file can be used by the Telecommunications program, TELCOM.

The kinds of information kept in the ADDRSS file ADRS.DO will generally include only name, address, and telephone number. However, like the scheduling function, additional identifying data may be recorded, such as:

P; Mom & Dad :4085551212: 2 Benjamin Court

S; Bob :6035551212: 144 Sheridan Street

D; Susan :6095551212: Winter Avenue

B; Charles Durang :6175551212: Reston

If you were to search this file for "Susan," then any occurrence of "Susan" would be shown, six at a time. However, if you were to detail all data applicable to your daughters, you would search for only "D;". Parents may be located by "P;". Business associates may be located by "B;", etc. Your coding system would simply be whatever works for your needs. If you desire to have the Telecommunications program (TELCOM) dial telephone numbers for you, surround that number with colons (:) and imbed no hyphens (-).

When you invoke the ADDRSS program, if the ADRS file has not been created, you will receive this message, accompanied by a beep from the computer's speaker:

```
ADRS.DO not found
Press space bar for MENU
```

If you receive the message, simply press the spacebar and you'll be back at the beginning. From that point it will be important to develop the ADRS.DO file using the TEXT processor. ADDRSS uses ADRS.DO, but doesn't create it.

When you enter ADDRSS, these three function keys will be initially assigned:

```
*                                              *
* Find                 Lfnd            Menu     *
************************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]      *
************************************************
```

Function Key functions are:

F1 (Find) permits a text search to be performed. It works the same as it does in the TEXT processor,

whereby you identify a name, date, classification, or other pertinent data from the ADRS.DO file.

F5 (Lfnd) works the same as F1, except that the located item is printed on a printer, rather than on the display. In this manner, when you indicate that no specific string is desired (you press ENTER), your schedule file will simply print out on the printer. You will prefer to use the PRINT command key with the SHIFT key to perform this function.

F8 (Menu) returns you to the opening menu of the application.

The Command Key functions are:

PASTE Allows inertion of text which had previously been COPYed or CUT. Thus, if you have been developing a second schedule, perhaps for the next month under another name, such as ADDRSS.DO, you could simply cut that portion of the second schedule and move it into ADRS.DO in this manner. This feature should be used frequently, as the smaller the ADRS.DO file is, the more efficiently the search mechanisms of ADDRSS will be, to say nothing of the amount of data to be displayed.

LABEL Displays the function keys used with ADDRSS. This key simply toggles the descriptive labels on and off. If toggled on, they occupy the bottom line of the screen. If toggled off, that line is available for file data.

PRINT Prints the display screen on the printer. When used with SHIFT, the entire file may be printed, which is the more efficient way to list the file.

BREAK Press BREAK (SHIFT/PAUSE) TO cancel operations.

The use of Find or Lfnd really produces only the next six occurrences of the search string. The operation must be repeated to locate the second set of six. Each occurrence may be the last, of course, but since there is no "find all" denotation of this function, it must be repeated. The MEW assists this requirement by displaying a second set of prompts.

```
*                                          *
*           More Quit                      *
*******************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]  *
*******************************************
```

Since more information may be displayed on the screen, F3 (More) will permit the next screen (6 lines) to be displayed or printed, depending upon whether you initially select Find or Lfnd. F4 (Quit) will return you to the initial display of function keys, where the process may be continued, or you may return to the main menu. That's all there is. It is no more complicated than that.

Telecommunications (TELCOM)

Unless something is to be communicated, there is little point to having a communications transmission capability. Therefore, we have concentrated on the development and manipulation of text data produced by the TEXT function of the Model 100. However, nothing described so far has any value without the ability to transfer it to another computer.

Of course, transmitting *to* is but half the function. Receiving *from* is the other half. This machine has the ability to gain information from another computer. And that other computer may be in the same room, directly connected through a null modem cable; or across the nation, connected by telephone lines. Connection to the telephone lines may be made in one of two manners: direct connection and acoustic coupler.

The purpose of the MEW is—must be—the ability to capture data which can be given to a more capable vehicle for further processing. In addition, the device must be an extension of that more capable vehicle. Thus, while a suitable document may be prepared for printing via the MEW, it's highly unlikely that its constant mode of operation will be word processing; and the SCHEDL and ADDRSS functions are merely convenience functions.

TELCOM has two modes of operation:

- **Entry Mode.** In this mode the machine may be used to dial a telephone number either for normal conversation or for computer-to-computer conversation, commonly called *communication*. When TELCOM is initially entered, it is already in Entry mode. The reasons for this are several, but let's simply say that this is a device which is manually controlled. No automatic dialing or automatic call receiving capabilities have been placed in this machine. What is called *automatic* dialing is really *semi-automatic* dialing, in that you initiate the search which finds the person record which initiates the call.
- **Terminal Mode.** In this mode the machine may function as a terminal to an information service or computer system. The terminal mode permits the travelling executive to receive electronic mail. It also permits a counter-oriented business to receive inventory values, enter inventory transactions to a more sophisticated computer host, and provide

a variety of very complicated solutions to business problems. The danger here is to think that permanent connection is impossible. It isn't impossible; though it may be highly improbable.

TELCOM EQUIPMENT

TELCOM assumes that you wish to communicate to another computer by one of these methods:

- Through a direct connection to the Bell System's Direct Digit Dialing system. The unit has a built-in modem which will handle the digital-to-analog translation. MODEM is an acronym for MOdulate/DEModulate. While this is not the place to teach communications theory, the computer is a digital device and the phone system is an analog service; the modem is required to convert each to the other. In any event, you will need a special cable known as the Model 100 Modem Cable (Radio Shack Part 26-1410). The cable connects to the DIN socket at the rear of the MEW. It has two leads. One lead connects to the modular connection mounted on the wall. The other connects the telephone handset. When not connected to the computer, the DIN plug is given a short-circuit socket so the telephone may continue to be used in a normal manner. When it is so connected, the DIR/ACP (direct/acoustic coupler) switch on the left side of the computer must be set to DIR.
- Through an acoustic coupler. Again, using the modem, transmission is done in a digital to analog fashion, and reception is the opposite. However, in situations where a direct connection is not possible, an acoustic connection is still possible. This is generally a function of older telephone handsets whose receiver is bulbous on both ends. You may obtain the Model 100 Acoustic Coupler. When connected, the DIR/ACP (direct/acoustic coupler) switch on the left side of the computer must be set to ACP.
- Through a null modem. The MEW is equipped with an RS-232 communications interface. This interface provides a plethora of services, but one of the more important services, at least insofar as the MEW is concerned, is its ability to directly transmit a document to another computer—and conversely to receive a document from another computer through direct connection. This requires the use of an RS-232 cable and a device known as a Null Modem Adapter (Radio Shack Part Number 26-1496). In effect, the Null Modem Adapter tells the computers that the connection will not be affected by communications restraints of the installed modem. A sex change adapter (Part 26-1495) may also be required.

ENTRY MODE

To invoke TELCOM, place the cursor on TELCOM in the menu and press the ENTER key. Then you are automatically in the Entry mode. LABEL has been activated, and Figure 3-2 shows what you will see on the screen.

```
*********************************************
*  M7I1E,10 pps                             *
*  Telcom:                                  *
*                                           *
*                                           *
*                                           *
*                                           *
* Find Call Stat Term                Menu   *
*********************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]   *
*********************************************
```

Figure 3-2

The function keys are:

F1 — F1 (Find) allows searching for names and phone numbers which had been stored in ADRS.DO. Press it and the word "Find" will appear beside the word "Telcom." Enter the search comparand, and then press ENTER. Thus, to call Reston, you would have only to enter "Reston," press ENTER, and the entire Reston entry will be shown.

F4 — F4 (Term) will put the machine into Terminal mode. Discussion of the Terminal mode will be included later in this chapter.

F8 — F8 (Menu) will return you to the main menu.

F3 — F3 (Stat) will display the communications protocol in use, as the communications protocol must match any computer with which it is communicating. The communications protocol shown previously M7I1E, 10 pps, means:

Baud Rate	M (300 baud)
Word Length	7 (bits)
Parity	I (ignore parity)
Stop Bit	1
Status	E (Enable)
Dial Pulse Rate	10 pulses per second

Naturally, we are not teaching telecommunications here. But we will detail the options available and state that if you know the "what," learning the "why" is purely a function of research. The factors must be known by TELCOM to use the system in Terminal mode.

F2

F2 (Call) is used after you have searched out the number you wish. In our case, searching could have been done on "Charles," "Durang," or "Reston," and the result is as shown:

```
*********************************************
*  M7I1E, 10 pps                            *
*  Telcom: Reston                           *
*  Charles Durang :6175551212: Reston       *
*                                           *
*                                           *
*      Call More Quit                       *
*********************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]   *
*********************************************
```

Figure 3-3

Now, pressing F2 (Call) will change the display to look like this:

```
*********************************************
*  M7I1E,10 pps                             *
*  Telcom: Reston                           *
*  Charles Durang  :6175551212: Reston      *
*  Calling Charles Durang :6175551212       *
*  Telcom:                                  *
*                                           *
*      Call More Quit                       *
*********************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]   *
*********************************************
```

Figure 3-4

The number being dialed will be displayed digit by digit. If you desire only to speak to an individual, pick up the phone while the number is being dialed. Press F3 to permit another call. Press F4 to return you to the previous screen.

TERMINAL MODE

The sole purpose of the Terminal mode is transmission of data from your computer to another computer or the receipt of data from another computer by your own computer. We assume that another computer will be accessed. The Model 100 must be placed on the ORIGination mode, TELCOM must be entered, and F4 (Term) must be pressed.

Connection to the computer communication service may be made via either manual or automatic dialing of the phone number. Once the high-pitched carrier tone of the service has been heard, the acoustic coupler must be connected (if necessary) or the phone set placed in its receiver (if by direct connection). At that time, the screen in Figure 3-5 will appear.

Figure 3-5

Automatic connection and *log on* to a system may be done using the Model 100. To *log on* is to provide identifying information to the communications service. In the case of the commercial service, this will be charge information. Password, or security, information will also be provided at this time.

The function keys which are assigned to TELCOM are:

F5 — F5 (Echo) is not automatically presented on the screen, but becomes active when pressed. It allows you to print information while it is being received. The transmission bogs down to printer speed, which may be of no concern. If not pressed, the received document will be held in memory and may be printed later. Depending upon the volatility of the received document, this function may not be used.

F6 — Immediately above F6 a "Wait" message may be seen. There is no function on this function key; that is, nothing will happen if it is pressed. The "Wait"

message tells you that the receiving computer cannot accept the data as fast as the Model 100 is sending it. Recall that the communications protocol had been "Enabled." At that time a connection was made. In telecommunications terms, it is called "XON." When the "Wait" message has been received, the corresponding computer has sent the "XOFF" code. When the backlog is cleared, the "XON" will be restored and the "Wait" message will disappear. Communications protocols will be discussed later in this chapter.

F1

F1 (Prev). Because data may be received faster than you can visually scan it, this key will permit you to back up and look at the previous screen. Pressing F1 will display that screen for you. Pressing it again will return the current eight lines. Things may often be too fast for this feature to be of significant value.

F2

F2 (Down) allows the incoming document to be saved. The process is called *downloading,* and is generally set up when the connection has been made. When pressed, the user is prompted for a name under which the new incoming document will be filed in memory. From that point the document will be modified or printed as necessary.

F3

F3 (Up) allows the transmission of a specific document across the communications media. The process is called *uploading,* and is set after the connection has been made. When prompted, you must specify the document to be sent (File to Upload?) and the Width of that document. Now, the specification of "communications media" is critical here, as the upload is a form of output. A serial printer, for example, may be connected to the RS-232 connector. Width then becomes critical. Similarly, an external modem or buffer may be connected to the RS-232, permitting transmission speeds of up to 19,200 baud, as opposed to the 300 baud which is standard with the built-in modem.

F4

F4 (Full) allows switching between Full and Half Duplex modes of operation. A Half Duplex service is typified by two-way communication, one end at a time. A Full Duplex service is typified by two-way communication, both ends concurrently. Most present day communications services operate Full Du-

plex, which permits the "echo" of the transmitted character to appear on the display.

F8 — F8 (Bye) exists the Terminal mode and hangs up the telephone line. When the key is pressed, the user is asked to confirm the desire to disconnect. A "No" response will leave the machine in Terminal mode. A "Yes" response will release the telephone line and return the user to the TELCOM screen.

COMMUNICATIONS PARAMETERS

Any communication which transpires from the MEW is a function of the modem. One of two methods may be used to transmit data—either directly through the modem or through the RS-232 communications adapter. To prepare the Model 100 for use as a communications device, the communications protocol must be established. The formula for the communications protocol is the five character string *rwpbs,* which stands for *r*ate, *w*ord, *p*arity, *b*its, and *s*tatus.

Rate, or specifically the *baud rate,* is a measure of speed in bits per second. These values may be selected:

```
M = (300, modem-selected)
1 =     75
2 =    110
3 =    300
4 =    600
5 =  1,200
6 =  2,400
7 =  4,800
8 =  9,600
9 = 19,200
```

Note that two codes have a value of 300 baud. The one which is coded "M" *selects* the direct connection or acoustic coupler through the modem, depending upon the setting of the DIR/ACP switch. The other codes bypass the modem and work with the RS-232 connector. The range of baud values permits matching the MEW to a variety of differently timed communications services.

Word, or more specifically *Word Length,* is a measure of the number of bits in the computer's character. Since the majority of current computers use the ASCII character set, the chances are that this value will be 7 (bits). *Bits* is a contraction for *binary digit;* the ASCII character set uses a 7-bit character. It's interesting to note that while a 7-bit character is used, an 8-bit character is permitted on most of the computers

in common use. Or, specifically, we speak of computer memories and processor chips in increments of 8 bits. The MEW is an 8-bit computer. The TRS-80, Model 16, is a 16-bit computer. The selection of bits, then, will be a function of the receiving computer. Likewise, some computers use other coding structures. For that reason, there are other word lengths. Permissible word lengths are 6 bits, 7 bits, and 8 bits.

Parity is a measure of consistency; the computer is constructed to transfer data to various places in memory or to various peripheral devices or services in a consistent manner. Since there are 8 bits in the character of the computer, and since there are 7 bits in the ASCII character, that leaves an extra bit, which is added or not added on the basis of the number of "1" bits in the character. Memory parity is customarily odd. Thus, if there is an even number of bits, then the computer adds a bit. Now, various peripheral devices require differences in parity. Parity codes will be E (Even), O (Odd), I (Ignored), or N (None). Communications services ignore parity. The parity bit still exists, but is not needed for communications. Communications are done *asynchronously,* or serially. This means that transmission of data is begun and ended by a stop bit.

The stop bit is added in the place of the parity bit at the end of the message. It is necessary to specify the number of stop bits. Most of the time a single stop bit will be sufficient. Certain protocols require two. Permissible codes are 1 (1 stop bit) and 2 (2 stop bits).

The Status code tells the computer on the other end of the line whether or not the transmission is interruptible. In communications parlance, it is known as XON/XOFF. The codes are E (Enable) and D (Disable). Then, as data are being transferred, the receiving computer can signal the MEW to send data or to hold up on the transmission. At that time, the "Wait" message appears above F6. If this code is set as Disable, then it will be necessary to manually send the codes. CTRL/Q provides the XON. CTRL/S provides the XOFF.

Finally, when the modem is used, a Dial Pulse Rate must be designated. Permissible rates are 10 and 20 pulses per second. The rate of 10 works with rotary telephone systems. The rate of 20 works with electronic telephone systems. And when the communications interface is invoked, those codes will be set with the Stat command, from the menu shown in Figure 3-6.

Once the Stat function key (F4) has been pushed, retype the communications protocol line. It will remain that way until it has been changed.

INVOKING TERMINAL MODE

The purpose of the Terminal mode is to transmit data from your computer to another computer or receive data from another computer on

```
*********************************************
*  M7I1E,10 pps                               *
*  Telcom: Stat                               *
*  M7I10,20 pps                               *
*  Telcom:                                    *
*                                             *
*                                             *
* Find Call Stat Term                 Menu    *
*********************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]     *
*********************************************
```

Figure 3-6

your computer. It is assumed that another computer will be accessed. The Model 100 must be placed on the ORIGination mode and the TELCOM must be entered.

MANUAL ENTRY

When you are not using the direct connection through the modular telephone connection, you must enter the Terminal mode manually. The same is true when you use an acoustic coupler.

You can place the call to the computer communication service either manually or by the automatic dialing of the phone number using TELCOM. Recall that when you enter TELCOM you are in the Entry mode. Generally when you're calling a host computer or bulletin board, the host will produce a high-pitched tone. Once that tone is heard, you must press TERM (F4) and hang up (if by direct connection) or connect the acoustic coupler. When you press TERM (F4), the MEW will generate its own high-pitched tone to indicate that the Terminal mode has been entered. Terminal mode may be entered automatically; we'll discuss that soon. Before that, let's deal with the next screen which will appear as shown in Figure 3-7.

```
*********************************************
*                                           *
*                                           *
*                                           *
*                                           *
* Prev Down  Up  Full Echo Wait       Bye   *
*********************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]   *
*********************************************
```

Figure 3-7

These are the functions:

F5 — F5 (Echo) is not automatically displayed on the screen, but becomes active when pressed. It allows you to print information while it is being received. It does mean that the transmission bogs down to printer speed, which may be of no concern. If not pressed, the received document will be held in memory. The received document may be printed later.

F6 — F6 (Wait) is not really a function key. The "Wait" message appears when the receiving computer cannot accept the data as fast as the Model 100 is sending it. Recall that the communications protocol had been "Enabled." At that time a connection was made. In telecommunications terms, it is called "XON." When the "Wait" message has been received, the corresponding computer has sent the "XOFF" code. When the backlog is cleared, the "XON" will be restored and the "Wait" message will disappear.

F1 — F1 (Prev) is a reverse scanner. Information incoming to your Model 100 comes fairly fast—and is displayed on the LCD in groups of eight lines only. If you missed something, the system provides the ability to back up to the *Prev*ious eight lines by pressing F1. Pressing it again will return the current eight lines.

F2 — F2 (Down) allows you to save the incoming document. The process is called downloading, and is generally set up when the connection has been made. When pressed, the user is prompted for a name under which the new incoming document may be filed in memory. From that point the document may be modified or printed as necessary. Think of it as receiving something "down" from a larger computer.

F3 — F3 (Up) allows the transmission of a specific document across the communications media. The process is called uploading, and is set after the connection has been made. To do so, specify the document to be sent (File to Upload?) and the Width of that document. Now, the specification of "communications media" is critical here, as the upload is a form of output. A serial printer, for example, may be connected to the RS-232 connector. Width then becomes critical. Similarly, an external modem or buffer may be connected to the RS-232, permitting transmission

speeds of up to 19,200 baud, as opposed to the 300 baud which is standard with the built-in modem. Think of it as sending something "up" to a larger computer. The communications protocol table previously presented will be in effect here.

F4 — F4 (Full) allows switching between Full and Half Duplex modes of operation. A Half Duplex service is typified by two-way communication, one end at a time. A Full Duplex service is typified by two-way communication, both ends concurrently. Most present communications services operate Full Duplex, which permits the "echo" of the transmitted character to appear on the display.

F8 — F8 (Bye) exits the Terminal mode and hangs up the telephone line. When the key is pressed, the user is asked to confirm the desire to disconnect. A "No" response will leave the machine in Terminal mode. A "Yes" response will release the telephone line and return the user to the TELCOM screen.

ENTERING TERMINAL MODE AUTOMATICALLY

With direct connection, you can invoke the Terminal mode on the heels of an automatic dialing procedure. To do so requires that you return to TEXT and make a change to the way the telephone number is filed in ADRS.DO. For instance, suppose that your local Bulletin Board (BB) is in your address file. You've filed the number (5551212) between the colons as before. To slip automatically into Terminal mode—that is, to avoid having to press Term (F4)—you have only to add the two characters "<" and ">" to the end of the phone number. Thus, the Bulletin Board number would be filed in ADRS.DO like this:

```
BB :5551212<>: Winchendon Computer Club
```

The steps to follow, then, are:

1. Set the ANS/ORIG switch (left side of the computer) to ORIG.
2. Enter TELCOM from the main menu.
3. Press Find (F1) and type BB, followed by ENTER. The code "BB" is the way to locate the number.
4. Press Call (F2), leaving the receiver on the hook.

If the connection is made, the MEW will then produce the high-pitched tone previously mentioned. If the service is busy, you will hear the busy signal through the computer. When the connections are properly made, you will then see the screen and function key display as described previously.

CREATING A LOG-ON SEQUENCE

Automatic connection and log on to a system may be done using the Model 100. To log is to provide identifying information to the communications service. In the case of the commercial service, this will no doubt be charge information. Password, or security, information will also be provided at this time. With the purchase of the MEW you receive a free hour at each of two services, CompuServe and Dow Jones. It is possible to tie into one of the services and to manually respond to its questions about user identification number and the password to be used. It's also possible to develop your entry into ADRS.DO to do it for you. Let's recall the last entry we made into ADRS.DO:

```
BB :5551212<>: Winchendon Computer Club
```

Suppose the Winchendon Computer Club placed a charge on its services, billed to you in CompuMoney, of course, and that it had a restriction solely to members. To enforce that, you would be assigned a membership number and a secret password. Your membership number would be, for example, "94966,123." And your secret password would be "BIT*BUCKET."

These two pieces of information would be placed into the calling sequence between the left and right carets (<>), along with a calling syntax. The calling syntax is extremely important *and is a function of the service itself.* To have the MEW act automatically, you must know what information the host service will send you and what your response must be. Therefore, let's first define the elements of the syntax, more specifically known as the TELCOM Auto Log-On Key Commands (see Figure 3-8).

Any character preceded by the caret symbol (^) is considered to be a control character, just as if the CTRL key had been pressed with the character. The caret is the shift mode of the number 6. Therefore to send a CTRL-C, the sequence is SHIFT/6/C. Some services require a CTRL-C to be sent first in the user-ID/password sequence. OK, let's change our bulletin board dial-up procedure to call automatically. Here is how the entry in ADRS.DO might look:

```
BB :5551212<=?D94966,123^M?PBIT*BUCKET>:
Winchendon Computer Club
```

TELCOM Auto Log-On Key Commands	
Key	Meaning
?	Wait for a specified character.
=	Pause for 2.0 seconds.
!	Send a specific character.
^	Causes the character which follows the ^ to be sent as a "control" character. The ENTER key is sent as ^M.

Figure 3-8

Reading from left to right:

BB	Your "Find" identifier for Bulletin Board.
:	Indicates that a telephone dialing sequence is about to begin. Must be paired on the other end of the sequence.
5551212	The telephone number of the service. Note that there are no hyphens. If the call were a long-distance call, the direct dial network interconnect number (1) and the area code must precede the number. Don't use this number, by the way, as all you will receive is Directory Assistance.
<	Automatically places the MEW into Terminal mode. Must be paired on the other end of the dialing sequence.
=	A 2-second pause to give time for the service to send its tone and for the connection to solidify.

Now, when the connection is made, the service will send this message: "User-ID:" and the cursor will expect you to respond. On the MEW side you must wait for the arrival of the message; however, you must select only one of the characters of the message before responding (the entire message could be specified). Therefore:

?D	Causes the MEW to wait until the character "D" of "User-ID" has been received before progressing.
94966,123^M	This is the user identification (fictitious, of course), followed by a CTRL-M, which is the same as the ENTER key.

Again the service responds, this time with "Password:", expecting that you will give it your secret code and press the ENTER key. Whereas before the last letter of the message was checked, the first letter is checked this time. There is no difference.

?P	Wait for the "P" of "Password."
BIT*BUCKET	This is the password, the secret code assigned only to you. In this instance, an asterisk (*) is imbedded, which is acceptable. It is possible, however, to use other characters. When that character is a "?" or an "=", then it is necessary to precede it with "!". Thus, had the password really been BIT?BUCKET, it would have been necessary to send it as BIT!?BUCKET, as the "!" directs the next character to be sent as is.
>:	These are the matching pairs which complete the automatic entry to Terminal mode (>) and which complete the telephone number specification for automatic dialing (:).

As you go through the sequence, Figure 3-9 illustrates what will appear on the screen.

```
*********************************************
*  M7I1E,10 pps                             *
*  Telcom: Find BB                          *
*  BB :5551212<=?D94966,123^M?PBIT*BUCKET>: *
*  Winchendon Computer Club                 *
*                                           *
*                                           *
*                                           *
```

Figure 3-9

When the call has been completed, the next screen sequence will be as shown in Figure 3-10.

```
*********************************************
*  User-ID: 94966,123                       *
*                                           *
*  Password: BIT*BUCKET                     *
*                                           *
* Prev Down  Up  Full                 Bye   *
*********************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]   *
*********************************************
```

Figure 3-10

And thus you are logged into the Winchendon Computer Club's Bulletin Board.

DOWNLOAD/UPLOAD

You're connected to the bulletin board, a menu of selections has been presented, you've selected one, looked at it, and decided you want a copy of it. You must download from the host computer. To do this, press "Down" (F2). Figure 3-11 shows what you'll see.

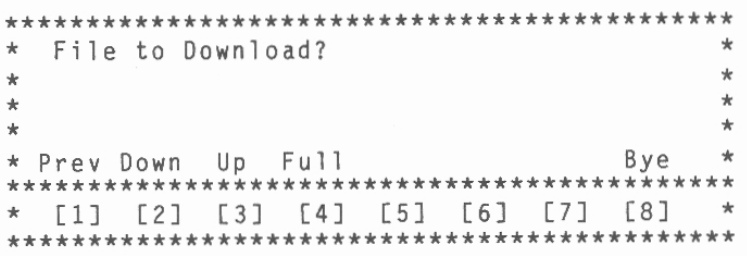


Figure 3-11

You must give it a file name (up to six characters) where that information will be stored, and follow with the ENTER key. During the process, the word "Down" will be shown in reverse video to inform you of what is going on. When transmission of the information has been completed, press F2 again to terminate the download function.

You can accept only as much information as you have room to receive. Thus, if you do this often, you can see the rationale for obtaining the maximum amount of memory. If there is more to be downloaded than you have space, the download will be automatically terminated when the available space has been used. Therefore, ensure that all unwanted documents and programs are either stored on cassette tape (and subsequently deleted from the MEW) or are simply deleted. Deletion is done via the BASIC kill procedure. Unfortunately, it is not possible to accept the first half of the document being downloaded, store it off, and then go back for the second half.

To log off and exit the Terminal mode, you must press F8 (Bye). You will be asked if you wish to disconnect; respond with a "Y" or "N" answer. "N" will abort the disconnect question and you may resume communications. "Y" will disconnect and hang up the phone for you. This will bring you back to the TELCOM menu. Press F8 (Menu)

again and return to the main menu. You can then examine the document using TEXT, manipulating or changing the information as you wish. Once you have edited the information, you may print it, again, according to the procedures which were outlined in the chapter on the TEXT processor.

Information may also be uploaded, or transmitted to a host computer. First, identify the document. When you have done that, it is necessary to identify the Width. The width of the MEW is 40 characters. If you exceed 132 characters (the largest print line in common use), the receiving service may break down your transmission into 132 character pieces. The purpose of the Width statement is to tell TELCOM where to place a carriage return and where to measure for wordwrap purposes. On most personal computers, a width of 80 is standard.

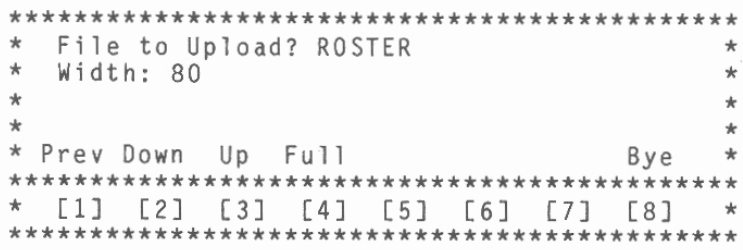


Figure 3-12

In Figure 3-12, the document being sent is ROSTER.DO, and it is being sent at a width of 80 (two MEW screen lines). Again, once the transmission is complete, to disconnect and get out of Terminal mode, you must press F8 (Bye). Again you will be asked if you wish to disconnect. To abort the question, press "N." To disconnect, answer "Y", and you're back at TELCOM. Press F8 (Menu) again and you will return to the main menu.

That's all there is to this thing called telecommunications—at least all that is required for the MEW owner. The next chapter will deal with the two services on which free time is achieved with the purchase of the device.

CHAPTER 4

On-Line Information Services

Recognizing that the executive is a mobile individual, Radio Shack has provided, with the Model 100, an hour's free access to each of two of the more prevalent on-line information services: CompuServe and Dow Jones. These are two of what are commonly known as the "big three" of general information databases. The third is known as "The Source." These are, by no means, the only on-line information services. There are several more, and while we will deal specifically with *how* to get into CompuServe and Dow Jones and some of the services which are specifically available with each, this first section of this chapter will describe those commercial services which are nationally available as of this writing. These do not include any local computerized bulletin boards.

The Big Three

CompuServe, Inc.
500 Arlington Center Boulevard
P.O. Box 20212
Columbus OH 43220
(800)-848-8199

CompuServe is said to be the most popular of the three networks. Located in Ohio, CompuServe is a subsidiary of the H&R Block organization, the same people who every year (whether you want them to or not) give you 17 reasons why they should prepare your income tax.

CompuServe's general database covers many topics. Business users may obtain news of their industry through CompuServe's financial analysis services. Nonbusiness users have several other services available, such as several home-oriented databases, an electronic mail system, and a CB radio emulator which permits intercomputer communication on an open channel.

CompuServe offers services in four major categories: Home Services, Business and Financial, Personal Computing, and Professional Services. Along with the usual business information on stocks, bonds, and commodities, and home services such as news, sports, shopping, banking, and games, CompuServe has such features as the Official Airlines Guide, an indispensable aid for the travelling executive. And, if you wish to learn more about computers, the service provides a periodical guide to microcomputing publications. The information is continuously updated. Subscribers have access to a variety of communications services. There are "talk" facilities and electronic mail. Likewise, word processing, data storage, advance programming languages, and specialized software are also available.

First time subscribers to CompuServe must purchase a starter kit, available at computer stores. That starter kit is part of your purchase of the Model 100, and includes one hour's "free" time. That free time, as will be shown, is an enticement to become a regular subscriber. In various stores the starter kit sells for $20 to $40. CompuServe gets $22.50 per hour prime time connect time (8 a.m. to 6 p.m., local time); if you wish to be a prime time user, there is a $45 minimum monthly charge. So-called "standard hours" cost $5 per connect hour during the 6 p.m. to 8 a.m. timeframe and includes no minimum charge.

Other charges for certain communications services and access charges for certain databases apply. Naturally, you must make financial arrangements in advance. CompuServe does not provide an 800 telephone number, but has local pickups at several major connecting points in each of the contiguous 48 states. In Massachusetts, for example, it has 15 pickups. California has 19. Utah has 1. Therefore you will be liable for any long distance charges between you and the nearest pickup point. According to CompuServe literature, the pickup list is expanding; you can obtain information from them at 800-848-8990 or (in Ohio) 614-457-8650.

Source Telecomputing Corporation
1616 Anderson Road
McLean VA 22102
(800)-336-3366

Next is The Source, a subsidiary of the *Reader's Digest*. It bills itself as an information utility, providing news reports, sports scores, weather reports, computer games, electronic mail, and mailgram generation capabilities. Boasting more than 800 databases in Communications, Business News and Services, Education and Careers, Government and Politics, Home and Leisure, News and Sports, Science and Technology, Travel, Dining and Entertainment, User Publishing, Creative Computing, and Consumer Shopping, entry fees and charges tend to run slightly higher.

The Source sees Dow Jones, rather than CompuServe, as its largest competitor. They recognize the viability of some of CompuServe's business, however, and provide electronic mail, computer conferencing, an open-line service called CHAT, and a service called Public, which is a bulletin board of sorts. The service operates 22 hours each day, being down from 4 a.m. to 6 a.m., EST. An initial fee of $100 is required to subscribe to The Source, plus hourly charges of $20.75 per connect hour (7 a.m. to 6 p.m.), $7.75 per connect hour (6 p.m. to midnight), and $5.75 from midnight to 7 a.m. There is a minimum monthly charge plus a charge of $.25 each time you sign on. You may have to hire an accountant to figure out your bill.

Dow Jones and Company, Inc.
P.O. Box 300
Princeton NJ 08540
(800)-257-5114

Dow Jones News/Retrieval is the largest of the on-line information services, with approximately 65,000 subscribers, as of this writing. It provides a wide variety of businss databases, general interest news, weather, sports, movie reviews, and an on-line encyclopedia. It has 18 databases in four major categories: Dow Jones Business and Economic News (with information from *The Wall Street Journal, Barron's,* and the Dow Jones News Service; Financial and Investment Services; Dow Jones Quotes; and General News and Information Services.

Databases are updated continuously and are on-line 22 hours daily (down from 4 a.m. to 6 a.m., EST), several days each week. Membership in the service costs $50. Prime time usage (6 a.m. to 6 p.m.) costs $54 to $72 per hour, depending on the database used. Non-prime time charges (6 p.m. to 6 a.m., weekdays) are $9 to $54 per hour. Access to the Dow Jones service is actually through one of two other communications networks: Tymnet or Telenet. Those charges are included in the quoted fees.

Again, introduction to Dow Jones services is part of the package you purchased with your Model 100.

All The Others (that we know about, anyway)

There is little point in detailing all the times, costs, services, features of all the other services available as of this writing. By the time this book is published, the number of vendors will have increased, the hours may have changed, the prices certainly will have changed. So we'll provide a listing, contact information, and a thumbnail description of the services provided by these other vendors in case one or more would provide a service useful to you. Contact the vendor directly—the author or the publisher of this book claim no responsibility for this information.

AUTO-CITE

The Lawyers Co-Operative Publishing Company
50 Broad Street
Rochester NY 14694
(800)-828-6373

A database of citations of legal cases and specialized reports in the areas of federal, regional, and state law, and special topics from the American Law Reports.

BRS SEARCH SERVICE

BRS
1200 Route 7
Latham NY 12110
(800)-833-4707

A bibliographic information service with 75 databases of text and abstracts of articles in Business/Finance, Medical, Scientific/Technical, Reference, and Education and Social Sciences.

BILLBOARD INFORMATION NETWORK

Billboard Publications Inc.
1515 Broadway
New York NY 10036
(212)-764-7424

Billboard music charts of 11 different types of music, including best-selling records in the United States. Includes industry-specific information for broadcasters.

BLOODSTOCK RESEARCH

Bloodstock Research Information Service, Inc.
801 Corporate Drive, Third Floor
P.O. Box 4097
Lexington KY 40544
(606)-223-444

Information on thoroughbred horses, pedigrees, race records, and stakes results. Information includes Broodmare Produce Records, Broodmare Sire Reports, Stallion Progeny Reports, Pedigrees, Domestic Stakes Results, Foreign Stakes Results, Thoroughbred Broodmare Sales, and Depreciation Schedules.

CAS ONLINE

Chemical Abstracts Service
2540 Olentangy River Road
P.O. Box 3012
Columbus OH 43202
(614)-421-6940

On-line information on chemical substances indexed by structural characteristics; sources include worldwide science and technology literature from 1965.

CHASE ECONOMETRICS

Chase Econometrics/Interactive Data Corporation
486 Totten Pond Road
Waltham MA 02154
(617)-895-4021

More than 120 economic and financial databases in the areas of U.S. Macroeconomics, U.S. Regional, International Services, industry-specific databases; and financial databases about stocks, bonds, commodities, and investments.

CISINETWORK

CISInetwork Corporation
16625 Saticoy Street
Van Nuys CA 91406
(213)-781-8221

Problem-solving services for corporations. Includes 14 databases in two categories (Economics and Energy) for organizations with United States and European interests.

CITISHARE INFORMATION SERVICE

Citishare
Box 1127
153 E. 53rd Street
New York NY 10043
(212)-572-9065

Economic statistics from more than 100 government and private sources: Standard and Poor's Compmark, Stockport, Compustat II, Insure, CitiBase, CitiData, CitiRate, and CitiQuote.

COMMODITY SYSTEMS

Commodity Systems, Inc.
200 W. Palmetto Park, Suite 200
Boca Raton FL 33432
(305)-392-8663

U.S. and foreign commodities information from 1922 to present. Stock information.

DELPHI

General Videotex Corporation
377 Putnam Avenue
Cambridge MA 02139
(617)-491-3393

Bulletin Boards, Conference, Banking, Games, Infomania, Library, Mail, News (UPI), On-line Markets, Travel, Personal Scheduling, and Word Processing.

DIALOG INFORMATION RETRIEVAL SERVICE

DIALOG Information Services, Inc.
3460 Hillview Avenue
Palo Alto CA 94304
(800)-227-1927; CA (800)-982-5838

Bibliographic information retrieval service with 175 databases in the areas of Business/Economics, Chemistry, Agriculture and Nutrition, Medicine and Biosciences, Energy and Environment, Science and Technology, Patents, Law and Government, Current Affairs, and Directories. Produces article abstracts.

GTE TELENET MEDICAL INFORMATION

GTE Telenet Communications Corporation
8229 Boone Boulevard
Vienna VA 22180
(703)-442-2500

Four databases of medical information maintained by the American Medical Association: Drug Information, Disease Information, Socioeconomic Bibliographic Information, and Medical Procedure Coding.

INFO GLOBE

The Globe and Mail
444 Front Street West
Toronto, Ontario, Canada M5V 2S9
(416)-585-5250

On-line edition of *The Globe and Mail,* the Canadian National Newspaper, and Marketscan (on-line stock market service).

ITT DIALCOM

ITT Dialcom, Inc.
1109 Spring Street
Silver Spring MD 20910
(301)-588-1572

National and international Financial, Medical, Governmental, Pharmaceutical, Energy, and Travel information.

KNOWLEDGE INDEX

Offered by DIALOG; see previous listing. Provides access to Standard and Poor's Corporate News, Business Information, Computers and Electronics, Engineering Literature Index, Agriculture (Agricola), Education, Government Publications, Medicine (Medline), Psychology, Magazines, Newsearch, National Newspaper Index (historical), Microcomputer Index, and International Software Database.

LEXIS/NEXIS

Mead Data Central
P.O. Box 933
Dayton OH 45401
(800)-543-6862

Lexis: legal research service; Nexis: full-text publications database. Includes *New York Times,* Shepard's Citations, Auto-Cite, Lexpat (U.S. Patents), Encyclopaedia Britannica, NAARS (National Automated Accounting Research System), AP Political Service, NY Times Information Bank, Deadline Data on World Affairs, Marketing Intelligence; abstract service from 160 publications.

MEDLARS

National Library of Medicine
Inquiries and Publications Management
8600 Rockville Pike
Bethesda MD 20209
(301)-496-6308

MEDLARS (MEDical Literature Analysis Retrieval System) contains 20 databases of medical information. Includes Medline, Citations from the *Biomedical Journal of Literature and Toxline* (drugs and chemicals).

M.J.K. DATA RETRIEVAL SERVICE

M. J. K. Associates
122 Saratoga Avenue, Suite 11
Santa Clara CA 95050
(408)-247-5102

On-line Commodity Computer Service database with information on commodity futures market (volumes; historical data; and opening, high, low, and closing prices) for private investors, brokers, and money managers.

NEWSNET

NewsNet
945 Haverford Road
Bryn Mawr PA 19010
(800)-345-1301 (PA 215-527-8030)

Three databases: NewsNet, NewsFlash, and Library, containing the full text of 100 electronic newsletters covering several industry areas.

NITE-LINE

National Computer Network
1929 N. Harlem Avenue
Chicago IL 60635
(312)-622-666

After-hours access to four financial and business databases: Comdat (commodities), Opdat (options), Fiscal (stocks and bonds), and Nite-Line News. Offers Media General Data Bank.

ON LINE RESEARCH

On Line Research, Inc.
151 Railroad Avenue
Greenwich CT 06830
(203)-661-1395

Four industry-specific databases in marketing, broadcasting, media, and advertising. Includes U.S. Census of Agriculture, U.S. Census of Retail Trade, Frost & Sullivan (market research), and Salary Grossups (tax computations).

ORBIT INFORMATION RETRIEVAL SYSTEM

SDC Search Service
2500 Colorado Avenue
Santa Monica CA 90406
(800)-421-7229 (CA 800-352-6689)

Reference service with 80 databases containing 55 million citations and abstracts from magazines, journals, books, patents, government reports, and other publications. Categories include Business, Chemistry, Engineering, Electronics, Energy, Environment, Government and Legislation, Industry-Specific, Life Science, Multidisciplinary, Patents, Social Sciences, and Science and Technology.

SECURITIES DATA COMPANY

Securities Data Company, Inc.
62 Williams Street, Sixth Floor
New York NY 10008
(212)-668-0940

Four databases on publicly offered and privately placed corporate securities, municipal debt issues of $5 million or greater, and merger and acquisition activity. Databases are: Mergers and Acquisitions, New Issues of Corporate Securities, Private Placements, and New Issues of Municipal Debt.

I. P. SHARP

I. P. Sharp Associates
2 First Canadian Place, Suite 1900
Toronto, Ontario, Canada M5X 1E3

More than 90 databases with a variety of business and financial topics. Includes: Aviation, Economics, Energy, Financial, Insurance, Banking, and Securities.

WEATHERSCAN

Weatherscan International
2445 Northwest 39th Expressway

Oklahoma City OK 73112
(405)-521-9230

On-line weather forecasts for specific geographic regions, includes Aviation, Agriculture, Municipal, Trucking, and Broadcasting.

WESTLAW

West Publishing Company
50 West Kellog Boulevard
Box 3526
St. Paul MN 55165

Legal cases, statutes, regulations for legal research. Federal Statutory Law, Administrative Law, Federal Case Law, Federal Special Interest Libraries, State Law Libraries, Shepard's Citations, Military Justice Reporter, Foreign Services Directory, and Tax Highlights.

This listing only scratches the surface. As carefully as possible we have researched and recorded the information in this listing. But new services emerge quickly.

CompuServe and Dow Jones Information Service

Packed with the MEW, at least as of this writing, are the access packets for both the CompuServe and Dow Jones Information Services. In each case you have purchased one hour of time on the service. While the information provided is not too difficult to cope with, an explanation will help. One procedural comment before beginning: If you call either the CompuServe or TYMNET services (the second is optional for CompuServe and required for Dow Jones) *and do not receive indication via the tone that you have made connection,* DO NOT BE ALARMED at the noises coming from the speaker of the MEW. Listen carefully as the pattern of noise may well indicate a busy signal. You will also be able to notice the effect of the busy signal on the display as the intensity alternately fades and brightens. Also, when you have a direct connection in use, be sure to plug in the line terminator plug when you have disconnected the plug from the computer. If you don't, people will wonder why your phone is always busy.

COMPUSERVE INFORMATION SERVICE

The "free" use of the CompuServe Information Service requires you to use the so-called Standard Service. Standard Service falls within the time period between 6 p.m. and 6 a.m., local time.

Communications Protocol

The communications protocol which is standard to the modem usage of the Model 100 is sufficient to communicate with CompuServe. Should you wish to communicate through the RS-232 connector, however, change the parity setting from (I)gnore to (E)ven. Transmission rates greater than 300 baud (standard in the direct connection modem) can be achieved with the RS-232 connector, but special arrangements must be made with CompuServe to accept those transmissions. Slower speeds than 300 are achievable through the RS-232, but if that rate is 110 baud (a commonly used teletype speed), a 2 stop-bit configuration must be specified.

Log-on Procedures

Of course, a modem is built into your MEW. The only question to answer is whether your connection with the phone system will be made via direct connection or acoustic coupler. The procedure varies slightly, but the information is the same.

Direct Connection. This is likely to be the more common method. The assumption is made here that your cables are properly connected to the modular phone jack, the computer, and to the telephone handset. The assumption is also made that you have properly stored the local CompuServe number, user identification number, and password in a syntactically proper record in ADRS.DO. In this example, the valid Worcester (MA) CompuServe number is given, but if the user identification and password shown actually work, it will be strictly coincidental. Figure 4-1 shows the automatic dialing information as it appears in ADRS.DO.

Ensure that the ANS/ORIG switch is set to ORIG. Place the cursor on TELCOM and press ENTER. Figure 4-2 illustrates the screen after we have entered TELCOM and pressed F1 to "Find the identifier CS".

```
*********************************************
*  CS :17930939<=^C?D94966,123^M?PBIT*      *
*  BUCKET^M>: CompuServe Information Svc.   *
*                                           *
* Find Load Save      Copy  Cut  Sel Menu   *
*********************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]   *
*********************************************
```

Figure 4-1

```
*********************************************
*  M7I1E,10 pps                             *
*  Telcom: Find CS                          *
*  CS :17939839<>                           *
*                                           *
*      Call More Quit                       *
*********************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]   *
*********************************************
```

Figure 4-2

From that point, the automatic log-on procedure does automatically what you would otherwise do manually.

Acoustic Connection. When using the acoustic connection, a few manual steps must be followed:

1. Place the cursor over TELCOM and press ENTER.
2. Ensure that the ANS/ORIG switch is set to ORIG.
3. Press F4 (Term) to put the MEW into Terminal mode.
4. Manually dial the CompuServe number in your area. For example in my area, the number is: 17939839.
5. When you hear a continuous, high-pitched tone, place the telephone handset in the acoustic coupler. The cups must fit the proper end of the handset.
6. Press CTRL-C. Hold down the CTRL key and press C, the required first character for many communications services, including CompuServe.
7. CompuServe will respond with a request for your user-ID number, which can be obtained from the user access packet. Answer this prompt with that number and press ENTER.
8. If the user-ID number is proper, then you are asked for the secret password they have given you. As you enter the password, be extremely careful, as you cannot see the characters as they

are entered. Those characters are not echoed to your machine, in case someone is looking over your shoulder.

And you'll end up with a screen display which looks like Figure 4-3 (nondisplayed information is bracketed).

```
*******************************************
*  User ID: 94966,123                     *
*                                         *
*  Password: [BIT*BUCKET]                 *
*                                         *
*                                         *
*******************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]  *
*******************************************
```

Figure 4-3

The one exception to the whole process is that, while CompuServe has many telephone pickups, they simply are not everywhere. They may be reached, however, through another national data communication service, TYMNET. To get to CompuServe through TYMNET you must dial TYMNET and get the high-pitched tone. You will then receive this message from TYMNET:

PLEASE TYPE YOUR TERMINAL IDENTIFIER

Your terminal identifier will probably be "A." TYMNET will then send the message:

PLEASE LOG IN

Respond with "CIS02" for nonprime time or "CPS01" for prime time. Follow either with the ENTER key. At this point you will receive the familiar CompuServe "User ID" prompt. Of course, if this sequence will be dialed automatically, all this information must be placed between the <>'s.

When you log-on to CompuServe you are given the opportunity to join the service as a regular subscriber. In increments, this is what your display will show *each time you log-on, if you have not acquiesced* (they use *your* time to advertise *their* service). The entire message is quoted here from the CompuServe literature, as published by Radio Shack, without the terminal breaks:

> Congratulations! You are now part of the exciting world of information provided by CompuServe. The Compuserve In-

formation Service (CIS) is continually growing and expanding and we urge you to log in regularly for news of the latest offerings.

Don't be afraid of pressing the wrong key. If you make a mistake, you are given a chance to try again.

CompuServe's Customer Service is ready to answer any questions not addressed by the CIS User Guide.

Your purchase of the CompuServe Information Service user's pack includes one free hour of use. To continue to use the service, you must open an account with CompuServe. Please read and understand the terms and conditions that apply to using this service. They are on the application included in your pack.

"To open an account you must both:

1. Provide CompuServe with your name, address and other information (you do this on the service by choosing to open an account).
2. Complete and return the application enclosed in your pack.

   "Your account with CompuServe may be paid through your VISA or Master Card or, for a monthly handling fee ($3.00), you may be billed directly by CompuServe. Do you wish to:

   1 Open an account
   2 Exit to the menu of services"

Hopefully, at least once in your experimentation you will select menu option 2. When you do, the whole gamut of CompuServe offerings becomes available. You have accessed a hierarchy. Your first entry into the CompuServe system provides what they call a *top menu*. It's called that because it's at the top of the hierarchy. From that top menu you will select a specific area. That area will lead you to the next lower level, which itself may be a Top Menu for a variety of services. The process is like a tree diagram, and you'll burn up your hour and then some looking for all the things available.

Summary of CompuServe Services

- News, weather, and sports from major newspapers and international news services.

- Financial information with updates and historical information on stocks, bonds, and mutual funds.
- Entertainment—theater, book, movie, and restaurant reviews plus information on opera, symphony, ballet, dance, museums, galleries, and more.
- Electronic Mail service where you can create, edit, send, and receive messages from any other CompuServe user nationwide.
- Home Information—a variety of government publications plus articles from home magazines.
- Personal Computing services including software exchange, line printer art gallery, challenging games, programming languages, word processing, business and educational programs, etc.

CompuServe Operational Details

Entry to CompuServe is standard entry. If you need to erase a character, erase it—assuming that you are making direct entry, which, of course, you will be while opening up the conversation.

Some control characters are pertinent to CompuServe operation:

^C	Interrupts the display or the execution of a program to permit entry to another menu selection or command.
^U	Deletes the current line.
^V	Redisplays the partial line you are typing.
^H	Backspaces, deleting the character.
^A	Suspends output at the end of the line.
^S	Suspends output immediately.
^Q	Resumes after ^A or ^S.
^O	Turns off the display of the output. Pressed again, it turns it back on.

It's not immediately obvious, but while you are working with your notebook-sized computer, CompuServe is out in Ohio in a very large computer center, populated with millions of dollars worth of machinery and a staff of highly qualified computer technicians and operators who must keep the whole thing working. Often it is necessary to shut the system down. It may be time for maintenance, or the system may be overloaded. Operators, then, may ask you to close things out, so while you're working away, there may come a message on your screen, preceded by ";;OPR." Usually when that happens, there is a planned de-

gradation of service, and you are given time to conclude your business, including any storage of information in one of their files (you are provided with 128,000 bytes of noncharged disk storage).

And then, occasionally, things fall apart all by themselves, and the network talks to you, as well. It may give you these messages:

? NTWCCN Cannot Connect.
? NTWLCP Lost Connect Path.
? NTWSPR System problem, please try again later.

If the message is preceded by "?" disconnect and try at a later time. If it is preceded by a "%" you may have to disconnect and try again, but the network is attempting to resolve the difficulty, e.g., % NTWCPR—Communication Problem.

You will no doubt be connecting to one of CompuServe's two main services: Videotex and Electronic Communication. Options provided by these two services include:

Videotex:

- What's New—Newly available CompuServe features.
- Using the Database—How to access database items and move from place to place in the Videotex area.
- Sending feedback to CompuServe—How to use the feedback service.
- Changing terminal or service type—How to change specifications.
- Current rates—What you'll have to pay today.
- CompuServe Subject Index—Displays the location (in page numbers) of the individual items in the database, by subject. The "G" command will allow you to select a specific page.
- CIS Command Summary—A brief summary of Videotex commands.
- CompuServe ViewPoint—Answers to questions people have posted on-line.
- Password Change—Required to change your access password.
- Summary of Month-to-Date charges—What your bill is.
- CIS access numbers—so you can select the one in your next city.

Electronic Communication:

There are three ways to send messages to other CompuServe subscribers and one with which to communicate with CompuServe Customer Services staff:

- CB—Like the citizen's band, this service permits direct communication between individuals. You have a multiple channel monitor and scanner, and a set of instructions as to how to go about it—10/4?
- EMAIL—This is a person-to-person message delivery system. You can read mail, send mail, receive information on undelivered mail, or file your mail for future reference. The process functions on system-assigned message numbers.
- BULLET—CompuServe's national bulletin board. On it you can create bulletins or read others' bulletins which have been placed there. Wait until you meet its editor, Filge.
- FEEDBK—Your questions for CompuServe. No charge is made for this service. You are asked for suggestions and problem reports. If you cannot log on, FEEDBK will be of no use to you. Contact them directly at 800-848-8990. In Ohio, call 614-457-8650.

DOW JONES NEWS RETRIEVAL SERVICE

First and foremost, if it is your desire to utilize the free hour of service from Dow Jones, you have 45 days from the date of the purchase of your MEW to do so. Dow Jones works a little differently from CompuServe as you must establish an account with them and gain the telephone number of the local TYMNET pickup from them (the telephone number of TYMNET is an unlisted number given only to subscribers).

Next in importance, you *must* change your communications protocol to use the TYMNET/Dow Jones configuration. Presently, you will recall, it is set for M7I1E, 10 pps. You must change that to M7I1D, 10 pps. The Enable feature (indicated by the "E") really enables the echo, which shows what you have sent back on your screen. TYMNET wants to put that information there all by itself, and requires that you disable (D) the echo. Recall from our discussion of TELCOM that this is done through F2 (Stat).

With your purchased materials you will find a Dow Jones Information packet. It is a sealed carbon set which looks like a contract for services, which, of course, it is. Fill out the blanks, except for the account number, which will be obtained presently. Sign it, and then break off the tear strip at the bottom. Near the bottom of the cardboard form you will find a User Identification Number which looks like Annnnnn, where "A" is a letter of the alphabet and "nnnnnn" is a six-digit number. That is not the user identification number which will be used when you are on-line, as with CompuServe. No, that number is the number which you will give to the customer services representative when you call 800-257-5114 (New Jersey and Canada—609-452-1511).

When they have obtained the same information from you that you will place on the form anyway (so they can send you an information packet), they will provide two items: an account number and a password. The account number will take the form: nAAAnnnn, where "AAA" are the first three letters of your last name. The password will take the form nnAAnnnZZZ, where "AA" are the last two letters of your last name and "ZZZ" are the three letters you have identified as being unique. Once you have recorded the account number on the form, tear off the cover sheet, fold it, seal it, and send it to Dow Jones News Retrieval Service, P.O. Box 300, Princeton NJ 08540.

After some considerable experimentation with the Dow Jones News Service, interacting manually with the system is easier than it is to build a solid automatic log-on sequence. This may be because you are really passing through two different services, in both directions, to achieve your results (TYMNET and DJNS). Therefore, it is recommended that you merely record the phone number and the Terminal mode connector (<>) in ADRS.DO and enter the rest manually.

When you have placed the call, and after the connection noise is detected, the sequence of events and responses you should give are:

1. Please type your terminal identifier. This is the TYMNET requirement. Respond with the letter A. Do Not Press the ENTER Key. TYMNET will respond with -nnnn-nnn-, which is an internal user number.
2. Please log in. In effect, you are being asked which of the available services on TYMNET you wish to use. Respond DOW1;;. The second semicolon functions as a carriage return and will not echo, i.e., it will not show on your screen. Therefore, do not press ENTER. However, when the terminal connection has been made, you will see the characters "tc>" on the right side of your screen display and the message "host is on line" will be at the left of the screen on the next line.
3. What service please? Type DJNS and press ENTER.
4. Enter password. Type the password you were given when you contacted the Dow Jones people—nnAAnnnZZZ. Press ENTER.
5. Enter query. From this point you're on your own. The user book plus the information packet will discuss the services you may receive.
6. When you are ready to disconnect, simply type DISC. The service will then respond with "please log in," for it anticipates that you may wish to come back. If you do not respond, the system will automatically log off in a couple of minutes, when the message "off line" appears. If you must use a long distance connection to get to TYMNET, then press F8 (Bye).

If you MUST have an automatic sequence, try this one (no guarantees):
DJ:pppppppp<=?fA=?GDOW1;;=?VDJNS^M?DnnAAnnn,AAA^M>:
where

DJ	is the "Find" identifier.
pppppppp	is the phone number, long distance.
A	is the terminal type.
DOW;;	is the log-on response.
DJNS	indicates Dow Jones News Service.
nnAAnnnAAA	is your password.

Pay particular attention to the wait characters (=) as TYMNET needs time to respond, since no carriage return character (ENTER key) has been received.

WHEN DJNS RETURNS . . .

Figure 4-4 shows what you'll see (or at least what I saw when I called them up).

```
*********************************************
*         DOW JONES NEWS/RETRIEVAL          *
*            COPYRIGHT (C) 1983             *
*         DOW JONES & COMPANY, INC.         *
*            ALL RIGHTS RESERVED            *
* MCENROE DOWNS LENDL TO MOVE INTO          *
* WIMBLETON TENNIS FINAL. SEE //SPORTS.     *
* TYPE //MENU FOR A DATA BASE LIST.         *
* ENTER QUERY                               *
*********************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]   *
*********************************************
```

Figure 4-4

The user's guide from Radio Shack is only 32 pages and doesn't contain the following information, available from page 19 of the user's guide Dow Jones will send: //INTRO is a free service, obtainable at any hour, not charged. So //INTRO is the QUERY we'll enter.

You don't have as many lines as this on your screen, but everything here will show and you can't slow it down. For purposes of illustration, the wrong code was entered to obtain the INVALID REQUEST

```
*********************************************
* WELCOME TO //INTRO, THE FREE NEWSLETTER   *
* OF NEWS/RETRIEVAL                         *
* PRESS       FOR                           *
* 1.  UNINET NOW AVAILABLE                  *
* 2.  1200-BAUD RATES INCREASED             *
* 3.  DATA BASE NEWS: FREE TIME             *
*     AVAILABLE IN NEW // FORBES            *
* 4.  POINTERS ... TRACKING MARKET          *
*     VOLUME AND MORE                       *
* 5.  WHO, WHAT, WHERE:  USAGE IDEAS        *
*     FOR SUBSCRIBERS                       *
* 6.  CUSTOMER INFORMATION:PRICES,          *
*     HOURS, USER AIDS                      *
* 7.  DOW JONES SOFTWARE NEWS               *
* INVALID REQUEST.  PLEASE PRESS            *
* T FOR TOP MENU, M FOR PREVIOUS            *
* MENU, OR P AND PAGE NUMBER FOR            *
* SPECIFIC PAGE.  FOLLOW ALL                *
* COMMANDS WITH RETURN KEY.                 *
* Prev Down  Up  Full                 Bye   *
*********************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]   *
*********************************************
```

Figure 4-5

message. Before you select //INTRO, press F2 (Down) and give the document a name so that you can store it. Then, when you have the information you want, you can sever your connection with the communications service (especially if you operate at long distance); then you can review the data—or print it off.

Learn where the reset button is on the back of your MEW by feel—it's the only way to disconnect the line under a busy signal condition. If the speaker doesn't indicate connect or busy, you may still have some crackling sounds on the line. These are the so-called *spurious emissions,* frequently called *glitches,* or *garbles.* If you have chosen to download the incoming document, these will show as strange characters in your text.

SUMMARY

If you know what Dow Jones can do for you, then the documentation which arrived with your purchase or which will be sent to you upon registration will contain the information you need. You can determine the charges which are available in prime (6 a.m. to 6 p.m.) and nonprime time (6 p.m. to 4 a.m., weekends and holidays) simply by accessing that database.

USPS Electronic Computer Originated Mail (E-COM)

This section would not be complete without some discussion of the U.S. Postal Service's Electronic Computer Originated Mail (E-COM). Touted as a viable alternative for the volume mailer who can generate mail from data stored in electronic form, E-COM is not for the individual who wishes to send one message. However, many who read this book could well consider generating mail using the MEW in E-COM format.

E-COM service is available in one of two manners. Because E-COM service is generally limited to mailers who can generate at least 200 messages per transmission to a Serving Post Office (SPO) within the 48 contiguous states, you can access the network directly only if you can generate that volume. If you cannot generate that volume, the USPS will provide you a list of common carrier companies who will consolidate your transmissions into a larger batch for forwarding to USPS.

Generally the service involves your transmission to one of the 25 SPOs, where they are printed on a capitals-only printer, folded, inserted into a Mailgram-appearing window envelope, and then carried in hard-copy form through the first class mail service. Each letter has a two page limit; one page costs $.26; two pages costs $.31. In addition, there is a $50 annual fee to establish and maintain the account, based on the anniversary date. Printing is done at 10 pitch, with a choice of six lines or nine lines to the inch (forms use nine lines).

Security is built into the system. The User Communication Subsystem (UCS) is capable of receiving data only. There is no way anyone can enter a code to gain access to your information. Only customers certified by the USPS will possess an access code to enter letters into the system. The customer identification code, when entered with the correct customer access code, will allow a certified customer to enter letters into the E-COM system. The UCS and the Main Processing Subsystem (MPS) are not hard-wired together. The SPO technician must physically remove the tape generated by customers from the UCS and move it over to the MPS, where the printing will be done, making it unavailable for further access. That particular technique is not the most modern amongst data processing methods.

To use the service you must establish the account with the Postal Service. Once established you have the option to transmit to any one of the 25 SPOs. Thus, if you have 200 or more pieces of mail to be transmitted from New York to San Francisco, your messages can be sent to San Francisco electronically and then delivered by first class mail thereafter. USPS guarantees two-day delivery, from a midnight cutoff, assuming local (to the SPO) delivery. Other deliveries will depend on normal First Class service. For example, transmissions received at the SPO in San Francisco on Monday night before midnight are guaranteed delivery not later than Wednesday's mail. Since volume mailers frequently

test the speed of the mails by mailing letters to themselves, you could conceivably send a letter round trip to yourself in four to five days. You are billed for the service from the USPS Management Operations Center (MOC), located in Wilkes-Barre, PA.

Three types of E-COM messages may be generated:

1. Single Address Message (SAM) includes the destination address and text required to produce a complete mail piece.
2. Common Text Message (COT) provides the capability to send identical letters to multiple addresses. A list of destination addresses must be provided in the submission.
3. Text Insertion Message (TIM) provides the capability to send letters of similar form to any number of addressees, with unique information fitted into a skeleton letter. The address list and unique information must be provided in the transmission.

These message types may contain any mix of ruled forms specifications and alphanumeric characters. The forms are specified in the Text Control block by a scheme of representing each ruled line by the row and column coordinates.

E-COM FACILITIES

The E-COM telecommunications connection arrangements will provide two broad classes of access facilities at each SPO: dial-up and dedicated. The computer capacity for handling access to E-COM serves both facilities equally.

Dial-up access facilities permit customers to connect to any SPO using public telephone numbers and comunications carriers of the individual's choice. Since dial-up is available to all, this provides the maximum access and flexibility, whereas dedicated access provides exclusive access to E-COM for those mailers whose volume is sufficiently large to justify the economics of the dedicated transmission facility. Generally the dedicated access facilities apply to the telecommunications common carriers and do not apply to you with your MEW.

DIAL-UP ACCESS FACILITIES

USPS will provide all of the telecommunications equipment and services at designated SPOs to permit dial-up access from any public telephone network, including communications interface protocol hardware

and software, modems, and network connections. Customers who wish to use this service will be provided with telephone numbers and transmission protocols for each dial-up telecommunication port. This data is available for the asking at your request: Ask for the *E-COM User's Technical Guide,* USPS Publication DM-501.

This information, complete with abbreviations and acronyms, was culled and quoted from the Customer Information Booklet (CIB) and the E-COM Promotional Brochure (E-COMPB, Publication 235). Included with the information booklet was a six-page list of telecommunications vendors (which you might use) and a three-page list of "softwear (sic) vendors" (which you probably will not need to use).

No doubt you'll be confused by the jargon, but let's simply say that the E-COM service will work with the Model 100. Read the directions for asynchronous ASCII transmission. Avoid the discussion of the binary synchronous protocols. Asynchronous transmission is done serially. Synchronous transmission is done in parallel.

If you know the binary numbering system, then the following example will have meaning. If you don't know the binary numbering system, this is not the place to teach it, and you'll have to accept the information on face value. The letter "A" has an ASCII value of 65 (decimal). In binary, the number 65 is represented as "1000001." Those values are assigned as follows:

```
64 32 16  8  4  2  1
 1  0  0  0  0  0  1
```

In asynchronous mode, these binary digits, or "bits," are transmitted across the communications line, one at a time. The rightmost "1" would go first, followed by five successive "0s" and then the leftmost "1." It would work something like this:

```
1 0 0 0 0 0 1 -->
```

In synchronous mode, each bit would be sent across its own communication line, and the whole character would be transmitted in parallel, like this:

```
1 -->
0 -->
0 -->
0 -->
0 -->
0 -->
1 -->
```

USER DATA FORMATS AND DATA SUBMISSION REQUIREMENTS

Data transmitted through E-COM must be *packeted*. That means that you must place your message in a package prior to transmitting it to the SPO. The "package" in this illustration has three parts:

1. E-COM Group Label—This information must include a type of character code (you will specify ASCII), a transmission group identification number, and an estimate of postage.
2. Message Blocks (any number of messages, in any mix of message types, each message consisting of the combination of header, text control block, text and address blocks). The exact format varies by letter type. These data blocks are used as appropriate:
   - E-COM Message Header—SAM (Single Address Message).
   - E-COM Message Header—TIM (Text Insertion Message).
   - E-COM Message Header—COT (Common Text Message).
   - Text Control Block.
   - Text Block.
   - Address Block.
3. E-COM End Block—Four bytes of data ($END) establish the end point of the transmission.

Details of the composition of each of these headers is available in the E-COM User's Guide.

TRANSMISSION PROTOCOL

Several types of protocol are available for communication with E-COM. That which applies to the MEW is the asynchronous protocol support, which uses the XON and XOFF characteristics described in the previous chapter. Line characteristics include Full Duplex mode, ASCII code, and the 300 baud transmission rate built into the modem incorporated into the MEW. If faster transmission is desired, and this would be a worthwhile effort if transmission from New York to San Francisco is a regular occurrence, you can reach other speeds and other protocols through the RS-232 connection on the MEW.

CERTIFICATION

USPS has not made it easy to use E-COM. If, indeed, you can wade through the information contained in their book, sooner or later you'll

come to grips with the fact that USPS will test your communications capability prior to taking you on as an account. Your interface equipment and procedures must be certified by successfully passing valid telecommunications traffic to the USPS Test and Development facility, which is equipped to analyze the test submission to assure trouble-free use of E-COM after certification. Read that not as your trouble with the system, but as their trouble with you.

Quoting from the promotional brochure—"Using E-COM service is simple . . . So why not call today?" It may well be simple, and certainly there are USPS representatives to assist, but it would be fair to state that the USPS system must become even more simple and more accessible to the individual or organization which generates a small volume before it is widely accepted as a public utility.

PART 2

TOOLS FOR THE PROGRAMMER

CHAPTER 5

Aw, Programming's Not So Tough!

We've now spent four chapters concentrating on things built into the MEW—all except the BASIC. By and large, if you wish to work only with the TEXT processor, the ADDReSS function, and the SCHEDuLe function, then you'll never need any discussion on the BASIC. However, two things are wrong with that line of reasoning: (1) sooner or later you will wish to do something which cannot be done only with those functions, and (2) your natural curiosity about how things work will push you further into this chapter.

Up until now you have worked with programs which have been permanently etched into Read Only Memory, or ROM. And, you have worked with the computer's external controls. You have learned that the external controls are the keys on the computer's keyboard. But a computer can operate under internal controls, as well. The internal controls are customarily embodied in a *program*. A program is a sequence of steps defined to the computer in advance so that the computer can perform them automatically. Although there are many ways to program a computer, the programming is usually the function of a computer *language*. The language of the MEW is BASIC. BASIC stands for *B*eginner's *A*ll-purpose *S*ymbolic *I*nstruction *C*ode, and the BASIC language is widely implemented on a plethora of microcomputers.

Every computer must have a means for accepting the entry of data—called *input*. The means available to the MEW take several forms, the most common being a keyboard. With a keyboard you can develop programs, build data files, respond to program commands, etc. Another means of entering programs or data to the MEW is the cassette drive. Because the MEW is designed to be a business machine, chances are

that certain business programs will become part of your fixed package, reloading from cassette only when necessary. Because of that, we'll concentrate largely upon the business activities.

Radio Shack's reference manual provides the somewhat technical explanations for the BASIC instructions. What it doesn't do is try to show how a program is constructed, and how to do it. We have already introduced a program (remember the alarm clock earlier in the book), and would like now to provide enough instruction to get you started in the programming activity. The purpose of this section is not to make competent programmers of our readers, but to whet your appetites.

Every computer has a place to store programs and information, called *memory*. The memory of the MEW is called a *chip* and is available to 32K memory sizes, as previously discussed. "K," you will recall, is equal to 1,024 positions of memory. Up until now, you have used memory only to hold the TEXT documents, i.e., name.DO, etc. However, memory is used to hold both programs and data. Further, a portion of memory is given over to *variables*, which are special positions of memory designed to contain numeric or alphabetic data.

In addition to the memory, the circuitry contains the electronic hardware that controls the *logic*, that is, the sequence of events that takes place within the computer. The logic section is part of the arithmetic/logic unit (ALU), which also contains the electronics for calculations.

Finally, every computer has some form of *output*. Output from the MEW takes the form of printed output (on a matrix printer), displays on the Liquid Crystal Display screen, or cassette. Other printers and devices may be attached to the unit as necessary, subject to design considerations.

Memory positions are technically called *bytes*. Each byte can be used to store a single alphabetic character (letters A to J), a single special character (punctuation marks and symbols), or a numeric value in the range of 0 to 255. Many of these bytes will contain the program you develop or purchase. Some will be used to contain the data upon which the program will operate. These data are called *variables*. For simplicity, we'll refer only to *numeric variables* and to *alphabetic variables*, often called *string variables*. Numeric variables are named with alphabetic names beginning with A and ending with J, and may be any reasonable length. However, the first two characters of the variable name must be unique. That is simply a requirement of the size of the system. String variables are named similarly, except that each name must end with a dollar sign—$. There are many other types and configurations of variables, but these two account for the majority you will encounter until you begin to use your MEW more extensively.

Numeric Variables

Let's concentrate first upon the numeric variable. Suppose, for instance, the numeric variables A to Z represent addresses of houses on a street, addresses that correspond to persons whose last names begin with one of those letters. Your name is Jorgensen and you live in house *J*. Mrs. Stevens customarily lives at house *S*, but is currently in the hospital. You have decided to collect money from your neighbors for a get-well present for Mrs. Stevens. Other people who live on the street are: Mr. Baker, who lives at house *B;* Mrs. Grant, who lives at house *G;* Mrs. Morton, who lives at house *M;* and Mrs. Richards, who lives at house *R*. Get the picture? We have labeled each house with a letter that serves as a symbol for its occupant and its address, and we'll treat each address as a repository of funds. For this example, your house and the total dollars you collect will *both* be given the name of the numeric variable *J*. You decide to kick the fund off with a contribution of $8. At this moment,

```
J = 8
```

In the computer, we would place the number 8—meaning 8 dollars—in the numeric variable J.

Mr. Baker chips in 10 lovely crisp dollar bills. Up to this point, those 10 dollars were located at B; that is, B = 10. But now, since Mr. Baker has given them to you, B = 0. (A zero is often slashed on a microcomputer, but we'll avoid doing so here).

Up to this point, all that was located at J was 8, but because Mr. Baker was generous, there is now $18 at your house. In other words,

```
J = J + 10
```

That equation might seem a little confusing. How can J be equal to itself plus 10? Well, the first letter J is known as the *result* or *sum*. Taking the original value of J, 8, and adding to it the 10 that represents Mr. Baker's contribution, provides a *new* J, or result, of 18. If we knew for sure that all the money Mr. Baker had was $10, then the equation could have been expressed as J = J + B. But we do not know that for certain. In our example, we will continue to compound the dollars at the single address J.

Technically, your original fund contained nothing—0. Thus, initially,

```
J = 0
```

To that you added $8, as follows:

```
J = J + 8
```

And then, Mr. Baker's $10:

```
J = J + 10
```

J now holds the number 18, which is the representation of the $18 you have acquired for the fund for Mrs. Stevens. Next, Mrs. White appears and she offers to double the amount you have already collected—two times $18, or $36. How would that be expressed on the computer?

```
J = J * 2
```

In BASIC, the asterisk (*) signifies multiplication. In this case, we took the value of J, 18, multiplied it by 2, and stored it again at J. J now has a new value of 36, representing $36 for the Stevens Fund. The use of the asterisk is required by computer convention to distinguish it from the "X" we use when calculating by hand—that X is one of the numeric variables.

Mrs. Stevens calls from the hospital to say she's just heard about the fund. Although appreciative of your efforts, she tells you that she's quite comfortable and doesn't really need the money. She asks you instead to distribute the money to her three favorite charities: the Muscular Dystrophy Fund, the Cystic Fibrosis Fund, and the Rolloff's Home. Furthermore, she advises that Mrs. Grant (house G), Mrs. Morton (house M), and Mrs. Richards (house R) are collecting, respectively, for those funds. How do we indicate that on the computer? Well, we know that Mrs. Grant is going to get one-third of the Stevens Fund for the Muscular Dystrophy Fund and that she lives at house G. We therefore divide our pile of dollars by three, as follows:

```
G = J / 3
```

In this instance, we took the values of the dollars at J, divided it by 3—the slash (/) is the computer's division symbol—and stored 12 at G. In other words, we said that G is equal to one-third of J, but just because we said so does not mean that we have taken the 12 away from J to give it to G. We must now tell the computer that the fund (J) is minus $12. That's done like this:

```
J = J - G
```

Why didn't we just say that J was equal to J minus 12? We could have done so in *this* instance, because we *know* the value of J. Suppose, however, that there had been other contributors. We needed to take merely a third of the Stevens Fund and that third is presently located at G. Thus, if the dollars at G are removed from the dollars at J, our totals will remain accurate.

Now $24 is left at J, half of which is to go to Mrs. Morton for the Cystic Fibrosis Fund and half to Mrs. Richards for the Rolloff's Home. The following instructions will place the funds in the proper place:

```
M = J / 2
R = J / 2
```

Of course, since nothing is left in your fund now, once again:

```
J = 0
```

Of course, there are many ways to obtain the desired result. We could have repeated the steps taken here and subtracted the variable J (J = J − M) and (J = J − R), but the result would have been the same.

On first examination, you'll note that we have created a series of very simple formulas. Programming the MEW need be no more complicated than this for any problem.

If you have bought this book *after* acquiring your Model 100 and understand the preceding chapters, you should already be familiar with the instructions provided, and no doubt the place you purchased the computer has shown you how to hook it up and get it started. Therefore we'll be concentrating largely on problem solving.

To invoke BASIC, move the cursor until it resides over the word BASIC, and press the <ENT> key. This is what you'll see:

```
*****************************************
*  TRS-80 Model 100 Software            *
*  Copr. 1983 Microsoft                 *
*  20083 Bytes free                     *
*  Ok                                   *
*  +                                    *
```

The specific number of "bytes free" will depend upon how much memory you purchase and what software is resident in memory at the time you use the BASIC. When it becomes necessary to return to the menu, simply press F8. The expression "Ok" is known as the BASIC prompt. The cursor shown is actually a flashing block of reverse video, located in the same position.

Now press the command key marked LABEL. Observe the bottom line of the screen:

```
*                                              *
* File Load Save Run  List            Menu    *
**********************************************
*  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]    *
**********************************************
```

The words on the line correspond to the numbers in the line immediately beneath. These numbers are painted onto the black frame of the display. These numbers, in turn, correspond to the eight function keys, marked F1–F8 in the left two groups of function keys. F1 corresponds to Files. Press F1. If you happen to have any documents, they will show on the display. Intrinsic programs will not be displayed. This allows you to see the available files without having to leave BASIC.

Using Save (F3), you can store a document on cassette tape (later enhancements may include disk). Load (F2) brings the document in from the cassette tape. List (F5) will list a BASIC program. Run (F4) will cause a BASIC program to operate. And you will recall that F8 is the standard return to the menu. Any of these can be typed at the BASIC prompt (Ok), at the location of the blinking cursor.

Stevens Fund Accounting

Let's walk through the Stevens Fund program we've just discussed to see what the computer will do with it. Type the following into your MEW:

```
J = 8 <ENT>
```

The <ENT> indicates that the ENTER key should be pressed. That will be continued throughout this portion of the exercise and then be abandoned.

What you typed is visible on the screen, but otherwise, nothing much seems to have happened. So try this next:

```
PRINT J <ENT>
```

In addition to the things you've typed, there now appears the number 8. What happened was that with J = 8, you set the variable J equal

to 8, and with PRINT J you instructed the computer to display the value of variable J on the screen, which it did. Now type this:

```
J = J + 10 <ENT>
```

Other than what you just typed, nothing else seems to have happened. So type this:

```
PRINT J <ENT>
```

The number 18 comes back. So far the program seems to be working just like the example. Type this:

```
J = J * 2 <ENT>
PRINT J <ENT>
```

If you did everything correctly, the answer 36 should now be displayed. To proceed, type this:

```
G = J / 3 <ENT>
PRINT G, J <ENT>
```

Here we've done something slightly different. We've set G equal to a third of the value of J and then displayed the values of both G and J. As you can see, G certainly equals 12, but J still contains 36. In other words, J is unchanged. That's why you should type the next instructions:

```
J = J - G <ENT>
PRINT G, J <ENT>
```

The variables should now be shown at their proper values. When you PRINT more than one variable, separated by a comma, the values of the variables will be "printed" (displayed) in different *zones*. There are two zones on the screen. To print the variables closer together, substitute a semicolon (;) for the comma, like this:

```
PRINT G; J <ENT>
```

See the difference? Now it's time to divide up the remainder of the Fund between Mrs. Morton and Mrs. Richards with the following instructions:

```
M = J / 2 <ENT>
R = J / 2 <ENT>
PRINT G; J; M; R <ENT>
```

If you did this correctly, the screen should show, left to right, the numbers 12, 24, 12, and 12, with a few spaces between. Now to "empty" J, type:

```
J = 0 <ENT>
PRINT G; J; M; R <ENT>
```

The screen should show 12, 0, 12, and 12. Does it?

Now, let's dress the display up a little bit by identifying who gets what, as follows:

```
PRINT "MRS. GRANT", G <ENT>
```

What happened? Did Mrs. Grant's name and the value 12 appear on the screen? They should have, because any time you PRINT something that is enclosed in quotation marks, it should appear on the screen. Be very careful about where you type the quote in relation to the comma. "MRS. GRANT," will produce an error because a comma before a quote is used to separate the *operands* of the PRINT instruction. If you used the comma correctly, Mrs. Grant's name will appear in zone 1 and the amount (variable G) in zone 2.

Up to this point, we've operated in Direct mode. You will occasionally hear this mode referred to as Console mode or as Calculator mode. The strength of the computer, however, lies in its ability to do things in the Indirect mode, which causes things to happen automatically. A program can therefore be developed to handle specific data even though the data are unknown until the program is operated.

Before we leave the Direct mode, however, let's try something else with the instructions:

```
J = ((G * 2) + M) / R <ENT>
PRINT J <ENT>
```

If you typed that into the computer and pressed the ENTER at the appropriate places, the computer will tell you how many people contributed to the fund. Although the formula is different from the earlier ones, you have done nothing different from what you've been doing all along. These formulas, by the way, are called *algorithms*. See how simple they are? The inserted parentheses group the actions to be taken in a specified sequence. In the computer's order of things, you have to multiply G by 2 before you can add M to it, and you have to add M before dividing the total by R. The parentheses specify the order of operation of an instruction within the computer. The inside parentheses must always be "cleared" first, followed by the next set.

Now try the operation this way:

PRINT ((G * 2) + M) / R <ENT>

It works just the same. To illustrate the importance of the parentheses, try the same formula without them, as follows:

PRINT G * 2 + M / R <ENT>

In the first instance, you came up with 3. In the second, you came up with 25. Do you know why? Type

PRINT G * 2 <ENT>

and you should get 24. Type

PRINT M / R <ENT>

and you should get 1. The sum of them, of course, is 25.

You could continue to operate in the Direct mode and cause the computer to produce answers for you. Suppose, however, that you want to display all the answers at once, rather than singly as they occur. Furthermore, suppose that you want a method whereby any contributor could contribute to any fund, which would then be redistributed to any charity of his or her choosing. In other words, the very concept of a program is that it can be operated automatically so long as you provide data for it, presenting the information you choose. Moreover, you want one computer to provide all those services.

A program is to a computer what a road map is to a traveller. It is a way to get from here to there, making the necessary stops along the way and making the necessary changes en route that can only be determined en route. To the computer, a program is the prespecification, i.e., thinking through, of a process you wish it to undertake. Like a note left for a friend, you detail the steps you'd like him to accomplish, advising him of alternatives, and instructing him in methods. And just like the note does, you number the steps. In BASIC, here is the short program that plots the route for the Stevens Fund:

```
10 J = 0
20 J = J + 10
30 J = J * 2
40 G = J / 3
50 J = J - G
60 M = J / 2
70 R = J / 2
80 J = 0
```

As each of the statements has been entered, you should have remembered to press the ENTER key.

Saving Your Program

Before long, you'll find that it's desirable to place your program into a tape file to protect it from loss or to obtain a paper copy. Before doing so, prepare a cassette to receive that information. Place a blank tape into the cassette tape unit and position it beyond the header. Ensure that the cables are properly connected from the MEW, as will be shown in the operator's manual, press the RECORD and PLAY buttons simultaneously, and then type:

```
CSAVE "filename" [,A]
```

There are a several ways to save a program to cassette tape. However, at this point, to save a program on cassette tape, you need only tell it to CSAVE, followed by a filename enclosed by quote marks ("). When you do that, the program is saved on cassette in a form called *Compressed mode*. The additional option "A" will cause the file to be saved in ASCII mode. ASCII mode is most useful if you wish to transmit the program across a communications media or to work with the program using the TEXT editor.

Experience using cassette tapes indicates that once that cassette unit is used for your computer, you'd better not use it for other purposes. Also, before you put anything critical on tape, practice with the tape unit to ensure that transfer to and from the tape works.

Reloading the program from cassette tape works similarly. The command is:

```
CLOAD "filename" [,R]
```

Naturally, the filename used must match the way it was recorded. The additional option, "R," will cause the program to RUN when loaded, making it unnecessary to press F4 (run). Again, there is another way to LOAD, and that will be discussed later. You can ensure that you have properly loaded your program by one of two actions—press F5. LIST, and the program will be listed on the screen, or

```
CLOAD? "filename"
```

will load the program again (make certain you have rewound the tape first), comparing what is on the tape to what is in memory.

Occasionally you may wish to clear memory of a program prior to loading or developing another. That's not difficult—simply type NEW. Follow it by LIST. Nothing should be there. NEW clears memory and LIST places any program on the screen.

Now let's talk about LIST. The program you've just written, CSAVEd, and CLOADed, can be shown on the screen with the LIST command. Remember that the keyword LIST is also available as Function Key 5. A program which isn't very long may show on the screen in its entirety. However, sooner or later you will develop a program that is much longer. Then, when you LIST it, the screen will continue to fill from the bottom while the program slips away at the top. This is known as *scrolling*. You can arrest the scroll simply by using the PAUSE key. But LIST has other capabilities:

LIST	Lists all statements.
LIST -100	Lists all statements thru 100.
LIST 100-200	Lists all statements from 100-200.
LIST 200-	Lists all statements from 200.

Make sure the program is in memory and type RUN (or press Function Key 4). This causes the program to begin execution. Although it looks as if nothing has happened, many things have happened *inside* the MEW. All the answers determined by the program now reside in the computer, but the computer hasn't told anyone about it. You didn't tell it to.

As you typed the instructions into the MEW, you may have wondered why you didn't number the steps 1, 2, 3, etc. You did, only you did it in multiples of 10. You could have used any number, so long as the next number was larger than its predecessor, but spacing of 10 units between instructions allows you to insert instructions later between the ones you've already written. For example, type the following instructions:

```
15 PRINT J
25 PRINT J
35 PRINT J
45 PRINT G,J
55 PRINT G,J
65 PRINT M,J
75 PRINT R,J
85 PRINT J
```

When you have completed entering these instructions, LIST them. If you wish to have a hardcopy listing, and you have connected a printer to the printer port at the rear of the machine, type LLIST to produce your paper listing. Word of caution here—Set-up for non-Radio Shack printers may be difficult. Radio Shack printers provide what is called an automatic linefeed/carriage-return (LF/CR), whereas other printers, such as the popular Epson MX-80 do not.

Your output should now look like this:

```
 8
18
36
12      36
12      24
12      24
12      24
 0
```

Well, you have your answers, but there is other information on the screen, and you don't really know, from looking at them, what those numbers really represent. To get rid of the extra information on the screen, insert the following instruction:

```
5 CLS
```

This is the "clear screen" instruction. It clears all the picture elements and moves the cursor to the upper-left corner of the LCD screen. Other ways to clear the screen will be discussed later.

Some of the ASCII codes are reserved to numbers and letters. Some of the ASCII values perform certain functions, and those functions may be invoked by the CHR$ instruction, which returns the value. The ASCII value 7, for instance, produces a tone when it is "printed," like this:

```
90 PRINT CHR$(7)
```

That tone is also available by:

```
90 BEEP
```

Add it to your program, for the effect.

RUN the program again. Did you hear the beep? We're going to change the instructions we added on page 97 to appear as follows:

```
15 PRINT " STEVENS:";J
25 PRINT " STEVENS:";J
35 PRINT " STEVENS:";J
45 PRINT "   GRANT:";G;TAB(20);"STEVENS:";J
55 PRINT "   GRANT:";G;TAB(20);"STEVENS:";J
65 PRINT "  MORTON:";G;TAB(20);"STEVENS:";J
75 PRINT "RICHARDS:";G;TAB(20);"STEVENS:";J
85 PRINT " STEVENS:";J
```

These numbers have been used before. When you retype them, they replace the old statements. The ability to do so is useful so long as

the instructions are improved, as they are in this case, but be careful not to rewrite an instruction unless you intend to. It is possible to *edit* these statements with an EDIT command, but that will be discussed a little later.

RUN the revised program. Doesn't it look better?

What you've done is to trace the progress of the Stevens Fund, the Cystic Fibrosis Fund, the Muscular Dystrophy Fund, and the Rolloff's Home. Generally speaking, we're interested only in the final results and not in the process and contents as we go along. The exception is when we must use the intermediate results to *debug* (i.e., check for errors) the program. So, let's get out the eraser. Since you can delete a line simply by typing its line number (followed by ENTER), type 5, 15, 25, 35, 45, 55, 65, 75, and 85. Was all that effort wasted? Not really, because you did get to see what was happening, and now you know some of the steps you must take to debug the program.

With that done, add the following instructions:

```
 90 PRINT " STEVENS:";J
100 PRINT "   GRANT:";G
110 PRINT "  MORTON:";M
120 PRINT "RICHARDS:";R
```

RUN the program again. There should now be four lines in the upper left hand corner of the screen, containing the name and amount of each fund. Add the following termination instruction (merely a branch to itself to freeze the display), and run the program again:

```
130 GOTO 130
```

You have now written and successfully run a computer program. The GOTO of the last statement is called an *unconditional branch* because it offers a means to change sequence without having to meet conditions. In this instance, the GOTO references the instruction that contains it. It keeps the computer operative, but in a "locked" mode.

Let's recap what we've done so far. We have talked about the Immediate or Calculator (Direct) mode and seen what it can do. We have talked about the Command (Indirect) mode. Recall that the Calculator mode functions precisely as a hand held electronic calculator. The Command mode includes all those actions at the keyboard that provide, among other things, the means to list a program on the screen.

By now you've discovered that your MEW can handle upper- and lower-case letters. Perhaps you've also discovered that any BASIC instruction typed in lower-case becomes upper-case after a LIST. We've learned that we can use letters to represent numbers or, more specifically, the value of numbers. The letters used (J, G, M, and R) are called

numeric variables. Actually, the setting of the numeric variable to a value is an abbreviated form of another BASIC instruction called LET. Statement 10 could have been written like this:

```
10 LET J = 8
```

The designers, in their wisdom, however, decided to let us express the instruction without having to type LET every time.

We've also learned that information can be displayed on the screen by using the PRINT instruction and that anything enclosed in quotation marks appears on the screen just as it has been typed. We've learned, moreover, that semicolons in the PRINT instruction cause numbers to be displayed in positions adjacent to other displayed information and that commas present information in zones. You'll discover also that if, instead of typing PRINT, you type ?, it will be converted to PRINT when a LIST is done.

And finally, we've learned that if we want to "short circuit" the instruction sequence, we have only to instruct the computer to GOTO some statement number. In the preceding example, we had the BASIC instruction 130 GOTO go to itself, thereby causing the computer to seem to stop. In reality, it continues to operate, but the operation is not obvious because it works inside the machine and is not apparent on the display.

REMarkable Program

Look at the list you produce on the display and at the hardcopy list if you prepared one. At this moment, you are fully aware of what you have accomplished. But in six months, after a few hundred more programs, you might have difficulty remembering what you originally tried to accomplish. Therefore it would be useful to be able to put comments in the program. Comments are entered with REMark statements, such as the following:

```
1 REM * MY FIRST PROGRAM *
```

The REM statement allows you to enter up to 255 characters of descriptive information (in fact, any instruction line cannot exceed 255 characters). REM statements may not include executable code. Good practice indicates that you should keep your REM statements confined to one line and use several lines, only if necessary. Further, while REMarks are both useful and necessary, they do occupy space and take

time to print. When memory space gets tight, the REMark statements are usually the first to be removed. An apostrophe may be used in lieu of REM. Use REMark statements judiciously—they take up memory, and memory is precious.

STOP, END, and CONTinue

When we wrote statement 130, we wrote it as a GOTO itself, causing the computer to continue to operate and to "hang." If you want the computer to come to a halt, replace statement 130 with either the command word END or the command word STOP. END, when used, will simply bring up the BASIC prompt Ok. STOP will produce a message: BREAK IN 130. STOP will also close any executing file, return BASIC to immediate execution, and redisplay the prompt. In either case—END or STOP—or when the BREAK combination has been hit, execution of the program may be resumed using the CONT command. The CONT command won't reopen any files and cannot be used to restart after an error. However, if there is no program in memory or the program cannot be restarted, you will be told that you cannot continue.

Making the Process Automatic

Given the fund raising problem and the values we have been using, the resulting answers will be precisely the same each time the program is run. Suppose, however, that we don't know the initial value of J and want to adapt the program to allow *any* value. Here's what we do:

```
10 INPUT J
```

Now, instead of giving J an initial value of 8, the computer will ask you what you'd like J to be. It will give a prompt, followed by a question mark, followed by the cursor (the position indicator on the screen). You must then enter the value that you'd like J to be. Note that the INPUT statement caused the computer to stop temporarily to await an answer. If you change statement 130 to GOTO 5, the computer will stop at statement 10 every time. Try doing so, entering different values for J. Note that as long as you enter a number, you will get an answer in return. Now try the same thing again, entering an alphabetic letter. You'll see that the computer will not permit this and will indicate an error.

The INPUT instruction is like PRINT because it allows a message to be displayed on the screen. Retype statement 10 as follows:

```
10 INPUT "ENTER CONTRIBUTION: ";J
```

and run the program again. You'll see that it takes the value you give it—J in this case—and returns the correct answer.

Determining the Size of Memory

Sometimes you wish to know how much memory is available. Each character on the screen that you have entered occupies one memory position, except for the line number. The line number is stored in binary and occupies two bytes only. Two bytes don't seem like much. However, when things are tight, instructions can be combined by placing colon separators (:) on the statement line, releasing one byte for each combination. The mandatory space after the line number accounts for a third byte, and that byte is also released when instructions are combined. Unfortunately, all the memory you purchased is not available to you. It includes overheads which must be paid. To determine the precise amount of available memory, type:

```
PRINT MAXRAM; HIMEM; FRE(0)
```

This will tell you, in sequence, the maximum RAM available to BASIC, the high memory address (which can be changed), and the available memory. Don't be alarmed if the numbers don't make sense—the only one which you'll really need to keep tabs upon will be the last, FRE(0), which will keep track of your diminishing memory resources.

Error Conditions

The MEW has 29 predetermined error conditions applicable to BASIC itself. You'll find those documented in the appendices of the Reference Manual. In addition, you can define your own error conditions and modify your program's direction accordingly.

Math Operators

In our little fund raising program we used some math operators, the symbols =, +, −, *, and /. Respectively, they indicate equation, addition, subtraction, multiplication, and division. The computer is, of course, a number machine, but it is decimally, not fractionally, oriented. When you begin to use or obtain very large or very small numbers, therefore, some strange things happen. The MEW uses *standard scientific notation* (also called *exponential notation,* or more simply *E nota-*

tion) in the presentation of very large or very small numbers. To illustrate, let's examine the number 8 in our decimal system. In the decimal system, the number is really 8. (i.e., eight followed by a decimal point, or 8.0). What would 8 million look like? It would look like 8000000. (Commas are not used to punctuate numerical data within a computer). That's eight, followed by six zeroes and a decimal point. In mathematical parlance, 8 million is:

$8 * 10 * 10 * 10 * 10 * 10 * 10$—or—$8 * 10^{+6}$

The MEW, like many computers, expresses very large numbers in E notation. The E notation does not commence until the scaling of 14 zeroes is encountered (this will be different for most computers), which would be a little much to demonstrate here. However, using the eight million as an illustration, in E notation it would then look like 8E+06. If the number gets too large, the MEW indicates overflow. That happens at 8E+61. Add one more zero, and overflow occurs. It's not likely that you'll encounter many numbers like this, but it's helpful to know what you're looking at. The important thing to remember is that counting begins at the decimal point.

Relational Operators

The computer is also a logic machine. It has a mechanical ability, or more specifically, an electronic ability, to reason. By *reason,* we mean the ability to make very simple decisions of the IF. . .THEN variety. In reasoning, the computer also uses its numbering abilities to compare numbers or characters to other numbers or characters. It can do so because each number or character has a numerical value or weight. The number 2 is "heavier," that is, larger, than the number 1. The letter B is "heavier" than the letter A. In the MEW, each letter is represented by a number. The number for the capital letter A is 65; for capital Z, 90; for small letter a, 97; for small letter z, 122. Thus, when sorted, small letters sort out after capital letters.

Suppose that we have two numeric variables, A and B. Both contain a number, but these numbers are not known. The numbers may be alike, or they may be dissimilar. Variable A may be greater than variable B (correspondingly, variable B is less than variable A). A may not be equal to B. B may be equal to or greater than A (which is, by definition, less than or equal to B). Get the picture? On the MEW, we express it by using the *relational operators* shown in Table 5-1.

These relational operators are customarily placed into an IF statement, like this:

Table 5-1. Relational Operators.

OPERATOR	RELATION TESTED	EXAMPLE
=	EQUALITY	A = B
<> OR ><	INEQUALITY	A <> B ; A >< B
<	LESS THAN	A < B
>	GREATER THAN	A > B
<= OR =<	LESS THAN OR EQUAL TO	A <= B ; A =< B
>= OR =>	GREATER THAN OR EQUAL TO	A >= B ; A => B

```
10 B = 8
20 IF B >= 6 THEN 50
```

In this instance, B contains the value 8, which is definitely greater than or equal to 6. The program sequence will then proceed to statement 50.

The use of the relational operators in an IF. . .THEN statement constitutes a *conditional branching* situation. The first variable (or number) following the IF is compared to the second variable (or number), and a branch is taken to the THEN address if the condition is met. The statement could have been written with the same results as follows:

```
20 IF 6 <= B THEN 50
```

Remember that the branching takes place conditionally upon the relationship of the first variable to the second variable, or, more specifically, the contents of those variables. If the branching to the THEN statement address is not taken, the computer assumes that the condition has not been met, and the program proceeds with the statement that follows statement 20.

In our fund raising program at statement 130, we told the computer to GOTO. GOTO is an example of an *unconditional* branch, that is, one that requires no conditional operators. The GOTO is customarily used to return to an instruction at the beginning of a process called *looping*. It's also used for changing direction within a program. Let's try some of these in a program:

```
5 CLS
10 A = 0
20 B = 5
30 C = 10
40 IF A > B THEN 90
50 PRINT "A not greater than B"
```

```
60 IF A <> B THEN 120
70 PRINT "A is equal to B"
80 GOTO 140
90 IF A = C THEN 160
100 PRINT "A is greater than B but less than C"
110 GOTO 140
120 IF A > B THEN 90
130 PRINT "A is less than B"
140 A = A + 1
150 GOTO 40
160 PRINT "A is equal to C"
170 END
```

This program exercises all but the greater than/equal combinations and thus indicates how conditional branching systems work. Enter this program into your MEW and run it. Your display should show the following:

```
A not greater than B
A is less than B
A not greater than B
A is less than B
A not greater than B
A is less than B
A not greater than B
A is less than B
A not greater than B
A is less than B
A not greater than B
A is equal to B
A is greater than B but less than C
A is greater than B but less than C
A is greater than B but less than C
A is greater than B but less than C
A is equal to C
```

Of course, on the MEW, the display will quickly scroll. You can cause it to PAUSE with the command key, or you can slow it down. To do so, add this statement at the end of every PRINT statement:

```
:GOSUB 180
```

The colon which precedes the command is very important. It is called an *instruction separator,* and permits adding another instruction to the same line. This instruction tells the computer to perform the instructions beginning at statement 180. At 180 we will code a *closed subroutine,* terminated by a RETURN statement. Purpose of the subroutine is to create a pause long enough for you to see what occurs. Here is statement 180:

```
180 FOR Z = 1 TO 500:NEXT Z:RETURN
```

Here we have a FOR. . .NEXT loop which will cycle a total of 500 times. In a FOR. . .NEXT loop, you must initialize the value (1) and indicate the maximum value (500). Those are two of the instructions at 180, separated by the colon. The third is the RETURN statement. The function of the RETURN statement is to return control to the instruction following the GOSUB. Thus, you will return to the next instruction following whichever GOSUB causes the closed subroutine to be invoked.

Now, just so you may see what has happened, the program will be modified slightly to present the variables as they are treated. Changes have been made in statements 50, 70, 100, and 160. Additions have been made in 55, 75, 105, 135, and 165, as shown here:

```
5 CLS
10 A = 0
20 B = 5
30 C = 10
40 IF A > B THEN 90
50 PRINT "A not greater than B"
55 PRINT TAB10);A;TAB(20);B;TAB(30);C:GOSUB 180
60 IF A <> B THEN 120
70 PRINT "A is equal to B"
75 PRINT TAB10);A;TAB(20);B;TAB(30);C:GOSUB 180
80 GOTO 140
90 IF A = C THEN 160
100 PRINT "A is greater than B but less than C"
105 PRINT TAB10);A;TAB(20);B;TAB(30);C:GOSUB 180
110 GOTO 140
120 IF A > B THEN 90
130 PRINT "A is less than B"
135 PRINT TAB10);A;TAB(20);B;TAB(30);C:GOSUB 180
140 A = A + 1
150 GOTO 40
160 PRINT "A is equal to C"
165 PRINT TAB10);A;TAB(20);B;TAB(30);C:GOSUB 180
170 END
180 FOR Z = 1 TO 500:NEXT Z:RETURN
```

Run the modified program, and the following program will result. Do not be concerned about the alignment of numbers in the final line. Because of the construction of the LCD, the values had to be brought to the second line; they were then tabulated for ease of viewing.

```
A not greater than B
        0     5     10
A is less than B
        0     5     10
A not greater than B
        1     5     10
```

```
A is less than B
         1     5     10
A not greater than B
         2     5     10
A is less than B
         2     5     10
A not greater than B
         3     5     10
A is less than B
         3     5     10
A not greater than B
         4     5     10
A is less than B
         4     5     10
A not greater than B
         5     5     10
A is equal to B
         5     5     10
A is greater than B but less than C
         6     5     10
A is greater than B but less than C
         7     5     10
A is greater than B but less than C
         8     5     10
A is greater than B but less than C
         9     5     10
A is equal to C
         10    5     10
```

The Editor

Previously you were asked to retype a line. That is always possible, as the address of the line—its number—directs replacement of the instruction if entered. Entire statements may be removed simply by typing the statement number and pressing <ENT>. If you wish only to change a statement, type EDIT, followed by the statement number. You will then be placed into TEXT mode and that statement will be shown. If you happen to try to EDIT a nonexistent statement you will be returned to the BASIC prompt Ok.

Once the statement has been displayed, move the cursor to the point where you wish either to add or delete information. If you wish to add to a statement, simply begin typing. If you wish to remove a character, press the SHIFT key and the BKSP key at the same time. That removes the character on which the cursor is resting. Press the BKSP key with the CTRL key and you will remove the character to the left of the cursor (backspace and rubout). When you have finished, if other statements are to be changed, move the cursor to the end of the line and press <ENT>. If you have completed editing, press F8 which will remove

you from the TEXT edit and return you to the BASIC mode. At this point you can exit to the main menu (F8), LIST (F5), or RUN (F4), or whatever else may interest you.

Renumber

Many computers have a renumbering function, where you can insert as many instructions as you need to and then renumber. Unfortunately, this is not true with the MEW. It wasn't true with the other Radio Shack models, either, but independent software developers derived renumbering programs which could be used on the hardware. No doubt by the time you are reading this book, someone will have such a program on the market.

If You Have Trouble

Since expertise in saving, loading, and other areas of programming comes with practice, you'll be an expert once you have practiced twice. But sooner or later, you'll have trouble. Dust or other contamination will do you in. Thus it's good practice to make backup copies of every file you produce. The best defense is a good offense. Make duplicates on cassette tape, transferred to larger computers, or printed.

If you do run into difficulty, particularly when using the cassette tape, make minor volume adjustments and try again. As the program loads, you can hear it in the MEW's speaker. Adjust for a solid and undistorted tone. Keep the recorder's heads clear. Keep those tapes away from dust, sunlight, electric motors (even battery-operated pencil sharpeners, I'm sad to say), or anything that could alter the magnetic characteristics of the cassette. If you choose to use cassette tapes, resist the urge to use inexpensive tape, i.e., cheap in quality. Use cheap tape if your interest is to keep an extra copy of SPACE TREKKIES. That's OK. However, don't even think of keeping anything important on anything but quality tape.

Loop Control

Recall our brief discussion of the FOR. . .NEXT loop. That loop was used to repeat a process. It happened to be a process leading to nothing, but the looping process is valid nonetheless. The number of times through the loop is controlled in this manner. That process is called *iteration*. In the FOR. . .NEXT loop, the pair of instructions surrounds the loop. For example, and to have a little fun, type:

```
10 FOR N = 12538 TO 415 STEP -100
20     SOUND N,1
30 NEXT N
```

The Model 100 has a capability to generate the tones of five complete musical octaves. Using the SOUND instruction that generates sound, beginning at the musical note F# (read that as "F Sharp" above middle C) and sliding upwards for five octaves. Format of the SOUND instruction is [pitch, length]. With a value of 50, the tone will sound for one second. In this example, the tone is set to two percent of that. The FOR. . .NEXT loop is established for a sliding value, with a decrement of −100 specified. That value falls within the range table for the SOUND instruction. Make the following changes to statement 20 so that you can see what took place.

```
10 FOR N = 12538 TO 415 STEP -100
20     SOUND N,1:PRINT N;
30 NEXT N
```

You could change the values to anything within that range; change the step; experiment. We'll talk more about music later. For now, our subject is loops.

Consider the following instruction sequence. The FOR instruction involves the use of a numeric variable (in this case S) and a range through which the count is to be made (in this case 0 to 59). It is possible to add a STEP value to the FOR instruction to change the method of calculation. At the moment, the FOR instruction is technically FOR S = 0 TO 59 STEP 1.

```
10 CLS
20 FOR S = 0 TO 59
30     PRINT S;
40 NEXT S
50 GOTO 10
```

Run the program. What happened? Oops! That's *movin'!* In fact, it's moving down the screen, clearing the screen, and continuing to move. It would continue forever unless you interrupt it with a BREAK or the power fails.

How could that possibly be useful? Who wants to count from 0 to 59 anyway? Anyone who wants to build a clock. Of course there is a built-in clock in the MEW and we do not want to replace that because it is automatic.

But it would be nice to understand how it works. You'll see that what we've set up is a nifty way to count seconds. Unfortunately, it's moving too fast to be accurate, and so we change the program to look like this:

```
10 CLS
20 FOR S = 0 TO 59
30     PRINT S;
35     FOR N = 1 TO 500:NEXT N
40 NEXT S
50 GOTO 10
```

Run the program again. What happened? It slowed down, didn't it? Break the program sequence with a BREAK and run the program again, this time with a sweep-second wristwatch in hand. Time the computer against the wristwatch. A new number should appear on the screen every second. If the hand on the watch runs faster than the computer, decrease the count at statement 35. If the hand on the watch runs slower than the computer, increase the count at statement 35. Note that the extra spacing in statements 30 and 35 are for readability.

How does the program work? First we set up a controlled loop that begins at 0, increments by 1 until variable S reaches 59, and then starts over again. That controlled loop looks like this:

```
20 FOR S = 0 TO 59
    . . .
    . . .
40 NEXT S
```

In other words, those instructions that fall between statement 20 and statement 40 will be done a total of 60 times (0 to 59). Embedded within the FOR . . . NEXT loop just mentioned is another FOR . . . NEXT loop that counts from 1 to 500. The MEW can perform approximately 500 single-step FOR . . . NEXT loops in one second. Thus, for every time through the "S" loop, the variable S is printed at a rate of approximately one per second.

Thus, the "N" loop has been made subordinate to the "S" loop. This procedure is called *nesting*. Loop "S" has a nested loop "N." Can that same principle be extended to the hour and minute hands? Certainly, but we'll have to start over. By the way, the addition of the semicolon to statement 30 merely allows continuous printing on the line. Delete it, run the program again, and watch it work without it. You can delete any instruction merely by typing the statement number and pressing the ENTER key. Unfortunately, the MEW does not have a DELETE command which will allow you to remove several instructions

(e.g., DELETE 10–40), but you can delete an entire program merely by typing NEW.

Now type the following. It is suggested that you observe the indentation, as the program will be more easily readable and traceable. While the spacing won't be exactly the same, you may use the TAB key to obtain the indentation. Note that the FOR. . .NEXT loops are paired:

```
10 CLS
20 FOR H = 0 TO 23
30     FOR M = 0 TO 59
40         FOR S = 0 TO 59
50             CLS
60             PRINT H; ":";M; ":";S
70             FOR N = 1 TO 500:NEXT N
80         NEXT S
90     NEXT M
100 NEXT H
110 GOTO 110
```

The MEW does have the ability to pinpoint a position on the screen, and we will deal with that later when we discuss the display screen.

This set-up may be very nice, but unless you stay up until midnight to run it, it won't keep accurate time, no matter how accurately you adjust the timing loop. If you take out the timing loop, you'll be able to see the whole thing function under test. How should the "clock" be set? Enter the following:

```
12 INPUT "ENTER HOUR";A
14 INPUT "ENTER MINUTE";B
16 INPUT "ENTER SECOND";C
```

Now edit statements 20, 30, and 40, substituting "A," "B," and "C" for the zero in each case, respectively. Also, calibrate the "clock" by changing statement 70, as previously discussed. Run the program and let it go for 5 minutes or so.

At the end of some time, look at your watch. Does the time indicated on the screen match your watch? If you made the entry at the beginning of the hour, it's probably close. But if you made your entry at 10 minutes to the hour or 10 seconds to the minute, you'll note that something is not quite right. Let the computer run another 10 or 15 minutes. When you come back, you'll note that things have gone from bad to worse. Why? Probably the hour indicator is still OK, because you haven't been gone that long. But no doubt the minutes indicator is approximately 15 more than it should be, and the seconds indicator is liable to be anywhere.

This has happened because a FOR. . .NEXT loop increments from the initial value you give it—which in this case was the our (A), minute (B), and second (C)—continues through until it finds the higher value, and then resets to the initial value that you gave it. If you had made your entry at 10 minutes before the hour (50), you'll "burn up" an hour every time you count to 10. Obviously, that's not right. We want that 59 at the second and minute indicators to revert to 0 even though we have entered something else at first. Here's what you should do to make this happen:

```
85 C = 0
95 B = 0
105 A = 0
```

Adjust the horizontal spaces on these instructions to make the symmetry correct, if you care to. Now run the program again, entering the correct time, and then check the time again in five minutes or so. It should be close. To calibrate the "clock" more closely, modify the count in statement 70.

Want to have a little fun? Try these changes:

```
20 FOR H = 23 TO A STEP -1
30      FOR 59 TO B STEP -1
40          FOR S = 59 TO C STEP -1
85          C = 59
95      B = 59
105 A = 23
```

Make the entries and run the program to see what it does, but first note the changes in the FOR statements. The controlled loop can be incremented (+) or decremented (−) by adding the STEP modifier discussed previously.

PRINTing to the Display Screen

PRINTing is really another way to say that a computer must have a means to produce output. Several forms of output are available to the MEW: hardcopy printout, cassette, and even communications lines. The form we must study, however, is the display on the output screen. (The terms *display screen* and *monitor* are interchangeable.)

The BASIC command PRINT sends output to the display screen. Its counterpart for hardcopy (paper) output is known as LPRINT (line

print). (LPRINT will be discussed later.) If you know that the entire contents of the screen constitute the sum total of your output, you may print the contents of the screen with the statement LCOPY. With LPRINT you generate one line of print. With LCOPY, you generate eight.

There are several ways to position information upon the screen. One of them has already been discussed: normal display (PRINT). The PRINT merely places its information in the next available position. Also, we've talked about PRINTing in contiguous spaces (those immediately adjacent to the previous position) and in zones.

We must distinguish between the *physical line* on the screen and the *logical line* to be displayed. In simple terms, a physical line is a fixed area of space on the face of the screen. There are eight physical lines on the face of the MEW's liquid crystal display. However, they are not numbered 1 to 8. The printing of characters is done in relation to the first position on the screen, which is zero. Thus:

```
Line 1 =   0 –  39
Line 2 =  40 –  79
Line 3 =  80 – 199
Line 4 = 120 – 159
Line 5 = 160 – 199
Line 6 = 200 – 239
Line 7 = 240 – 279
Line 8 = 280 – 319
```

So long as you address the specific position on the face of the screen, you will be able to position characters at a predetermined location. Later we will learn how graphics are positioned. The method is different, but the concept is the same.

The following will demonstrate the positioning of the letter "X" on the entire first four lines of the display:

```
10 CLS
20 FOR N = 0 TO 159
30     PRINT @N,"X"
40 NEXT N
```

Note that the cursor placement is dynamic, that is, it depends upon the value of the variable N, as modified by the FOR. . .NEXT loop.

The responsibility for positioning of printing is yours. Perhaps the simplest way to develop a screen is to develop an array which will be subsequently printed on the screen. You will not understand everything shown here, but this is one way to develop a screen:

```
10 CLS
20 DIM SC$(8)
30 FOR N = 1 TO 8
40     SC$(N) = STR$(N)
50 NEXT N
60 FOR N = 1 TO 7
70     PRINT SC$(N)
80 NEXT N
90 PRINT SC$(8);.
100 GOTO 100
```

Statement 20 DIMensions an array called SC$ (for SCREEN$) and designates it will have eight elements. You must ensure that each element never exceeds 40 characters (really 39, as a linefeed is added).

Statements 30 to 50 simply load the array with the string value of the number from the FOR. . .NEXT loop. We'll talk about STR$ later, but let's simply say that the array had been defined as string variable (the $), and N is a numeric variable. The STR$ converts the numeric value to a string value.

The linefeed (which occurs on a normal PRINT statement) will shift the line upward when the eighth line has been filled, beginning with a clear screen. That eighth line shifts upward, leaving a blank line and scrolling the top line off the screen. Thus, seven of the lines are printed as a function of the FOR. . .NEXT loop, while the eighth is printed separately, and is ended with a semicolon, which is the way the linefeed is suppressed.

Statement 100 freezes the display, else the BASIC prompt would be displayed, shifting the entire screen display upward. Enter the program and try it.

Therefore, for a message to appear properly on the screen, its presentation must be planned. For the most part, messages should be fewer than 40 characters, and to avoid a scroll, terminate the PRINT instruction by a semicolon.

If the semicolon is used, the cursor will be placed at the next position immediately following the printed message. Should you simply PRINT, you would begin at that position. This, then, makes spacing considerations important when combining messages. Examine the following two instructions. Enter them, run them, and then add a semicolon to the end of statement 10 and run them again.

```
10 PRINT "This statement contains";"an error."
20 PRINT "This statement contains";" no errors."
```

As can be seen, the instruction in statement 10 will create an erroneous message, which will nevertheless be displayed. To correct the message, embed a space before the second part of the message, as in

statement 20. The displayed output from these instructions will look like this:

```
This statement containsan error.
This statement contains no errors.
```

The spacing of mixed lines is extremely important. Look back at the PRINT line used in the clock example. It was carefully constructed as PRINT H;":";M;":";S. If you took a good look at the screen while that example was working, you may have noted that spaces surrounded the numeric variables. This space is customarily reserved for a sign, and positive signs (+) are not displayed. In this instance, the colon is enclosed within the quotation marks, meaning that it is a part of the data to be displayed. If it were between two instructions on the same line, it would function as an *instruction separator*. Insofar as BASIC instructions are concerned, spaces must surround numbers. Spaces surrounding operands are not always mandatory, but are used for readability.

The completion of a PRINT instruction forces a carriage return (linefeed). Ordinarily, any PRINT instruction following a PRINT instruction with carriage return will display on the next available line. However, if the PRINT instruction is terminated by a semicolon, the carriage return will be suppressed. For example, the instructions

```
10 PRINT "Printing this way ";
20 PRINT "allows you to ";
30 PRINT "break up the ";
40 PRINT "instruction without ";
50 PRINT "breaking up ";
60 PRINT "the line."
```

produce output that looks like this on the screen:

```
Printing this way allows you to break up
 the instruction without breaking up the
 line.
```

That same line would print on a single line on the hardcopy printer. Now a word of caution. Adding a printed line to a previously printed line is strictly contingent upon the ability, spacewise, to fit the entire next message on the same display line. Observe statements 50 and 60 and then observe the printed output or display upon the screen. The addition of the words in statements 50 and 60 is too much for a defined screen width. Although you can enter an instruction with as many as three lines of characters, and the line will "wrap" as needed, the same is not true of printed output.

If you type PRINT with nothing following, you'll get only a linefeed, giving you a blank line. The shorthand notation for the PRINT instruction is to type a ? (question mark) under a numbered statement. LIST it and you'll observe that it has been translated to PRINT. This is because the ASCII token for the PRINT instruction is the question mark. What's a token? Each of the instructions in BASIC is reduced to a single character by the interpreter. That single character is known as a token.

If you follow the PRINT statement with a variable expression (A, B, C, etc.), the values of those variables will be displayed contiguously if separated by a semicolon or in one of the MEW's two zones if separated by a comma. If you wish to use a string expression, you may use either a string variable (A$, B$, C$, etc.) or enclose the string with the PRINT instruction itself, between quotation marks. If spacing for several messages is required and you find that the previously described methods don't work, there are still a couple of ways to accomplish it—embedded spaces or tabulation.

Before we look at tabulation, however, change the PRINT instructions to LPRINT instructions (if you have a printer), and run the program again. You'll find that it works precisely the same way on the printer, except that the printer has a width of 80.

Tabulation

One interesting variation of the PRINT command is the ability to structure the screen into columns easily. It could be structured into zones, of course, but this approach limits you in presenting information on the screen. That problem is solved by the TAB subsection of the PRINT (and LPRINT) command. Consider the following instructions:

```
10 PRINT TAB(1);"CHECK";TAB(10);"PERSON";
20 PRINT TAB(20);"AMOUNT";TAB(30);"DATE OF"
30 PRINT "NUMBER";TAB(11);"PAID";
40 PRINT TAB(21);"PAID";TAB(31);"CHECK"
```

If you enter these statements and then run them, you'll see headings line up as follows:

```
 CHECK      PERSON     AMOUNT     DATE OF
NUMBER       PAID       PAID       CHECK
```

Then, in successive lines, you can align the data to be presented in the same manner. Remember two important things about TAB:

1. It is a separate operation in the PRINT command and *must* be both preceded and followed by a semicolon, unless specified first after the PRINT statement.
2. It is descriptive of a position in the physical line itself, with a range of 1 to 80.

Blinking Display

The message displayed upon the screen can be made to blink. You must establish the display in its appropriate place, print the message, pause for a period of time so that it can be read, clear the message, pause for a similar period of time to provide relief, and repeat the process. Here is a program to do this:

```
10 CLS
20 A$ = "THIS MESSAGE BLINKS ON AND OFF"
30 B$ = "                                "
40 PRINT @40,C$
50 FOR Z = 1 TO 200:NEXT Z
60 IF S = 1 THEN C$ = B$:S = 0:GOTO 40
70 IF S = 0 THEN C$ = A$:S = 1:GOTO 40
```

Let's discuss the technique used in this routine. Note that statements 20 and 30 are identically sized; one contains the message and the other contains blanks. Statements 60 and 70 comprise a small switching arrangement. In the first, if the switch is *on* (variable S set to 1), then the message is cleared and the switch is turned off. In the second, if the switch is *off* (variable S is set to 0), then the message is loaded and the switch is turned on. It's important to note that in each case, the right hand instructions in their respective three-instruction sets are executed only if the condition of the first instruction is met. In other words, if S = 0 at statement 60, then control would immediately transfer to statement 60.

The PRINT @ statement merely places the message at the beginning of the second line on the screen. Statement 50 provides the delay, else the message would flash too rapidly to be seen.

The Printer

Insofar as the presentation of text is concerned, use of a hardcopy printer is essentially the same as the use of the display itself. Instead of PRINT, however, the instruction used is LPRINT. Rules for tabulation and spacing are the same.

PRINT @ does not apply to printers, but a whole host of other items do, and before we can begin to study those things that are specific to printers, we will have to introduce a new instruction (and one that we've used before)—CHR$, or string character.

CHR$

As previously stated, the MEW is designed to operate in ASCII. Let's examine the character set that can be displayed on the screen. Enter and run the following:

```
10 CLS
20 FOR N = 0 TO 12
30     PRINT N;CHR$(N);
40 NEXT N
50 PRINT 13, CHR$(13)
60 FOR N = 14 TO 255
70     PRINT N;CHR$(N);
80 NEXT N
```

This routine shows you the character set. Because CHR$(13) is the same as CLS, it has been isolated. While these codes are displayed, you may wish to touch the PAUSE command key.

Many of the ASCII codes mean different things to the printer. Some of them are printable graphics characters. Some are control codes for spacing; we'll discuss these next.

This program will display the graphic codes on the printer. The Radio Shack DWP-410 letter quality printer contains some French, German, and Japanese characters. Try the routine on your own printer. If you have a matrix printer, you should see some graphic characters.

```
10 CLS
20 FOR X = 1 TO 16
30      LPRINT X+159;"  ";CHR$(X+159);
40      LPRINT TAB(10);X+159+16;CHR$(X+159+16);
50      LPRINT TAB(20);X+159+32;CHR$(X+159+32);
60      LPRINT TAB(30);X+159+48;CHR$(X+159+48)
65      LPRINT
70 NEXT X
80 END
```

With the Epson MX-80 printer, for example, there are a few differences that can be made to your printed output through the use of specialized ASCII characters and CHR$ codes. The same is no doubt true with other types of printers. Refer to the ASCII charts in the MEW literature and with any printer you may have.

These codes work with the MX-80:

CHR$(7)	sounds a horn at the printer when used in an LPRINT statement. It sounds an alarm within the system unit when used in a PRINT statement.
CHR$(14)	sets the print to double size (and therefore takes up double the horizontal space on the paper). Note that the vertical spacing does not change.
CHR$(15)	sets the print size to smaller than normal size with a compression that allows 132 characters to be printed within the width normally used by 80 characters.
CHR$(18)	turns off the compressed mode.
CHR$(20)	turns off the double size mode. These two mode changes must be used when returning from the selected size to the normal size of print, which is 10 characters to the inch.
CHR$(27)	is the code of ESCape. This code must be used preceding those CHR$ codes that follow in this list, as the ESCape code clears the buffer. A buffer is a temporary location for data or programs between memory and the diskette drive or between memory and the printer.
CHR$(69)	turns on an emphasized mode of print. Since ASCII 69 is also the letter "E," that letter may be used when preceded by CHR$(27).
CHR$(70)	turns off the emphasized mode. Since ASCII 70 is also the letter "F," that letter may be used when preceded by CHR$(27).
CHR$(71)	turns on the double strike mode of print. ASCII 71 is also the letter "G," enabling that letter to be used when preceded by CHR$(27).
CHR$(72)	turns off the double strike mode. Since ASCII 72 is also the letter "H," that letter may be used when preceded by CHR$(27).

If you have a matrix printer, enter and try this program:

```
10 LPRINT CHR$(14);"      TITLE LINE"
20 LPRINT
30 LPRINT "Normal Line"
40 LPRINT
50 LPRINT CHR$(15);"This line is in condensed print"
60 LPRINT
```

```
70 LPRINT CHR$(18);"Normal again"
80 LPRINT
90 LPRINT CHR$(27);"E";"This line is emphasized print"
100 LPRINT
110 LPRINT CHR$(27);"F";"Back to normal again"
120 LPRINT
130 LPRINT CHR$(27);"G";"This is double strike print"
140 LPRINT
150 LPRINT CHR$(27);"H";"Back to normal again"
160 END
```

Run it and you will achieve the different types of printing indicated in the comments.

The Print Mask

If you wish to align a column of figures on the screen or on the printed output, use a command known as PRINT USING. To do so, a device which is commonly called the *print mask* must be used. The print mask is a filter through which a numeric or alphabetic string can be edited. Concentrating first upon numeric strings, editing refers to the insertion of decimal points, commas, dollar signs, algebraic signs, floating asterisks, etc. Consider the following program:

```
10 CLS
20 INPUT "ENTER A NUMBER: ";A
30 PRINT "THE NUMBER YOU ENTERED WAS:";
40 PRINT USING "$$,###.##";A
50 PRINT
60 INPUT "ENTER NUMBER,COMMA,NUMBER";A,B
70 PRINT "NUMBERS ENTERED: ";
80 PRINT USING "$##.##     ";A,B
90 PRINT:PRINT
100 END
```

This is its output:

```
ENTER A NUMBER: 1234.56
THE NUMBER YOU ENTERED WAS: $1,234.56

ENTER NUMBERS,COMMA,NUMBER: 12.34,56
NUMBERS ENTERED: $12.34   $56.00

ENTER A NUMBER: .23678
THE NUMBER YOU ENTERED WAS: $    0.24

ENTER TWO NUMBERS, SEPARATED BY A COMMA: .14,1234.56
NUMBERS ENTERED: $ 0.14     $%1234.56
```

Compare the output to the program. Note specifically the differences in the INPUT statements at 20 and 60. In the first instance, only one variable is requested (A), whereas in the second, two variables (A and B) are requested. Note the punctuation that precedes variable A. In some computers, you have the choice of a semicolon (which produces a question mark) and a comma (which does not). The semicolon is standard on the Model 100.

Now about the mask. The first mask ($$,###.##) is set up to provide a sliding dollar (double) sign, punctuation, and space for six numbers (including one now containing the $). Note that when 1234.56 was entered, it was edited perfectly as $1,234.56. In the second example, two numbers are accepted by the INPUT statement and placed appropriately relative to the decimal point. Note that in the final entry, the number was too large to be edited. It was still placed in the output, but flagged with a % to let you know it could not be edited by the printing mask you provided.

Finally, look at the edit mask in line 80. The spacing between the numbers on the display is a function of the extra spaces contained in the mask variable. If you place a semicolon between A and B in this statement, you will be told that it is erroneous. The comma will display the numbers directly adjacent unless you include the spaces in the mask.

These same masks function with LPRINT, and you'll no doubt have to experiment to get them right since myriad usable combinations exist. These are detailed fully under PRINT USING in the BASIC section of your Model 100 manual.

GOTO

It often happens that you must change direction within a program. You may wish to change direction unconditionally; that's done with GOTO. You may wish to change direction with anticipation of returning; that's done with GOSUB. Or you may wish to do either on the basis of a selection of values; that's done with the instructions ON...GOTO and ON...GOSUB.

The GOTO is straightforward. You've completed a loop (examine several of the programs we've discussed) and you wish to repeat it. The GOTO statement will unconditionally branch to the statement you want.

You might want to develop an amount of instruction coding to be used over and over again but you don't want to code it each time it is needed. At such times you can use a subroutine. Subroutines are any routine that is terminated with a RETURN command, as shown in the following program:

```
10 CLS
20 PRINT "PERFORM A SUBROUTINE"
30 FOR N = 1 TO 500:NEXT N
40 GOSUB 1000
50 GOTO 10
1000 '****************************************
1010 '*                                      *
1020 '*         THIS IS A SUBROUTINE         *
1030 '*                                      *
1040 '****************************************
1050 CLS
1060 PRINT "DISPLAY A MESSAGE"
1070 FOR N = 1 TO 500:NEXT N
1080 RETURN
```

Although this routine simply exchanges messages on the screen, it clearly demonstrates the use of the subroutine. To get into a subroutine, you need a GOSUB. To get back to the next sequential instruction, a RETURN must be used at the end of the subroutine.

When your responses are in a defined order, it's possible to place a multiple branch within the same statement. For instance, suppose that you had begun your program with a menu, or a selection of options. Enter the following program and run it:

```
10 CLS
20 PRINT "1.    MENU OPTION ONE"
30 PRINT "2.    MENU OPTION TWO"
40 PRINT "3.    MENU OPTION THREE"
50 PRINT "4.    MENU OPTION FOUR"
60 PRINT
70 INPUT "SELECT: ";A
80 ON A GOTO 100,120,140,160
90 GOTO 10
100 PRINT:PRINT "YOU SELECTED MENU OPTION ONE"
110 END
120 PRINT:PRINT "YOU SELECTED MENU OPTION TWO"
130 END
140 PRINT:PRINT "YOU SELECTED MENU OPTION THREE"
150 END
160 PRINT:PRINT "YOU SELECTED MENU OPTION FOUR"
170 END
```

As you can see, if you select one of the offered options (follow your entry with the RETURN), this option will be displayed. Try one that is not offered, and you'll see that the selected option is rejected. This is called *error trapping*.

The same result can be achieved with GOSUB. Modify the preceding program as follows:

```
10 CLS
20 PRINT "1.     MENU OPTION ONE"
30 PRINT "2.     MENU OPTION TWO"
40 PRINT "3.     MENU OPTION THREE"
50 PRINT "4.     MENU OPTION FOUR"
60 PRINT
70 INPUT "SELECT: ";A
80 ON A GOSUB 100,130,160,190
90 GOTO 10
100 PRINT:PRINT "YOU SELECTED MENU OPTION ONE"
110 GOSUB 220
120 RETURN
130 PRINT:PRINT "YOU SELECTED MENU OPTION TWO"
140 GOSUB 220
150 RETURN
160 PRINT:PRINT "YOU SELECTED MENU OPTION THREE"
170 GOSUB 220
180 RETURN
190 PRINT:PRINT "YOU SELECTED MENU OPTION FOUR"
200 GOSUB 220
210 RETURN
220 FOR N = 1 TO 1000:NEXT N:RETURN
```

One minor twist occurs here. Note that within each separate subroutine, a common subroutine is executed—a subroutine within a subroutine. The latter subroutine is known as a *nested subroutine*.

INKEY$

To enter data without having to press the <ENT> key use INKEY$. INKEY$ is the computer's method of strobing the keyboard for the entry of a single alphanumeric character. Since the ON...GOTO and ON...GOSUB statements require a numeric quantity, you must convert the alphanumeric value being received from the keyboard to a numeric value useful to the ON. A more sophisticated way to do this exists but the following method works and is more easily understood:

```
10 CLS
20 PRINT "1.     MENU OPTION ONE"
30 PRINT "2.     MENU OPTION TWO"
40 PRINT "3.     MENU OPTION THREE"
50 PRINT "4.     MENU OPTION FOUR"
60 PRINT
70 PRINT "SELECT: ";
80 Z$ = INKEY$:IF Z$ = "" THEN 80
90 IF Z$ = "1" THEN A = 1:GOTO 150
100 IF Z$ = "2" THEN A = 2:GOTO 150
110 IF Z$ = "3" THEN A = 3:GOTO 150
120 IF Z$ = "4" THEN A = 4:GOTO 150
130 GOSUB 300
```

```
140 GOTO 10
150 PRINT A
160 ON A GOSUB 180,210,240,270
170 GOTO 10
180 PRINT:PRINT "YOU SELECTED MENU OPTION ONE"
190 GOSUB 350
200 RETURN
210 PRINT:`BINt "YOU SELECTED MENU OPTION TWO"
220 GOSUB 350
230 RETURN
240 PRINT:PRINT "YOU SELECTED MENU OPTION THREE"
250 GOSUB 350
260 RETURN
270 PRINT:PRINT "YOU SELECTED MENU OPTION FOUR"
280 GOSUB 350
290 RETURN
300 PRINT
320 PRINT "INVALID SELECTION -- RE-ENTER"
330 GOSUB 350
340 RETURN
350 FOR N = 1 TO 1000:NEXT N:RETURN
```

Most of this program should be straightforward. About the only thing we haven't discussed is the double quote (" ") at statement 80. Known as a *null string,* it merely redirects the program back to the keyboard strobe (INKEY$) when no entry has been made.

As previously remarked, there is a more sophisticated way to do this. Study the instructions between statements 90 and 150 in the program, and then consider how this instruction might replace them:

```
90 IF VAL(Z$) < 1 OR VAL(Z$) > 4 THEN 160 ELSE 10
```

SUMMARY

Let's take stock of how far we have come. We'll do so by reviewing the commands, statements, and functions we have already learned. Some new instructions have been added which will be useful. In addition, you may use this section for easy reference in the future.

COMMANDS

EDIT n — Enters the EDIT mode by the command followed by a statement number. Move the cursor to the point of change. Insert by typing. Delete the current cursor location by pressing SHIFT/BKSP. Delete the pre-

	vious character by pressing CTRL/BKSP. Return to BASIC program by pressing F8.
FILE	Advises what files are stored within the RAM memory of the MEW. Use Function Key 1.
LIST (n–m)	LIST by itself—without the (n–m)—will display a program on the screen. Once begun, the program, if it is too long for the screen, will scroll. The scroll may be arrested by the PAUSE/BREAK key and restarted in the same manner. If only a part of a program is required, the inclusive lines can be specified with LIST (n–m). Also, n or m can be separately specified (using the hyphen) to select only a portion of the program for display. Display of the entire program may be achieved by pressing the function key assigned to LIST (F5).
LLIST (n–m)	Same as LIST but targeted to the line printer; the same rules apply.
LOAD"filename	Used to load a program from a cassette. The BASIC keyword is available as Function Key 2.
NEW	Wipes out the program in memory and clears all variables.
RUN	Causes the program in memory to be executed. If the program is loaded from cassette tape and the RUN option follows the filename, the computer will read the program from the tape and begin execution. Available as a Function Key (4).
SAVE"filename	Used to record a copy of the program located in memory onto a cassette tape. The BASIC keyword is available as Function Key 3.

STATEMENTS

END	Returns the program to the command level. The "Ok" prompt reappears.
FOR. . .	Repeats program lines a specific number of times. The ellipsis can consist of a variable, expression, target expression, or incremental step. FOR ... is used with NEXT.
GOSUB	Transfers control to a subroutine, which must include a RETURN statement.
GOTO	Unconditionally branches to a specified line.

IF	Used to test logical conditions, allowing branches at logical differences.
LET	Assigns the value of an expression to a variable. Not necessary to use.
NEXT	The closure for a FOR...NEXT loop.
ON...GOSUB	Performs a subroutine selected on the basis of the value of a variable.
ON...GOTO	Unconditionally branches to a selected subroutine on the basis of the value of a variable.
REM	Includes REMarks within a program. This function can also be generated by an apostrophe.
RETURN	Is used to return from a subroutine. It enables a return to a particular location, given the line number.
STOP/CONT	Stops the program execution, prints the break message, and returns a program to command level. The program can be restarted with a CONTinue command.

INPUT/OUTPUT STATEMENTS

CLS	Clears the screen and puts the cursor in home (upper left) position.
INPUT	Reads data (either alphabetic or numeric) from the keyboard.
LPRINT LPRINT USING	Prints data on the printer. If editing is desired, a print mask can be developed for punctuation, dollar signs, etc.
PRINT PRINT USING	Displays data on the display screen. If editing is desired, a print mask can be developed for punctuation, dollar signs, etc.
SOUND	Is used to generate a musical note within five octaves, establishing the frequency and duration.

STRING FUNCTIONS

CHR$	Displays on the screen or the line printer, any ASCII character within the range of 0–255. The same ASCII character may generate entirely different display characteristics on the LCD display and the printer.

TAB(n)	Used for horizontal space on display or printer.
VAL(x$)	Used to determine the value of a number that has been developed in alphabetic form.

There are others, of course, and we'll get to most of them in later chapters.

CHAPTER 6

Developing and Manipulating Data

Now that we understand how the MEW functions, we can turn our attention to the development of programs, routines, and methods of handling data. Data, of course, are the raison d'etre for possessing a computer. The program within the computer must work upon data that are either (1) already contained within the computer, (2) entered from outside the computer, or (3) developed as a result of previous manipulations on either or both types of data.

Thus far, we have used certain types of data—namely numeric variables and alphabetic variables—the latter being signified by the terminating dollar sign on the name (label) of the data. A variable may be thought of as a specific place in which to store things, a place where size and type constraints are assigned to the data. The MEW has a limitation in the size of its name—the first two characters are the significant characters. Variable names may be larger, but characters beyond the second position are not treated as relevant. Variable names customarily begin with a letter. Thereafter, any of the available letters, digits, or characters may be used. The names may be totally unique in and of themselves, but if the first two characters are the same, they will all refer to the same variable. ANN, ANDY, and ANDERSON would all refer to the variable AN. Likewise, because of the MEW's limitations, the variable name may not contain a BASIC keyword. For example, neither ANDY or ANDERSON would have been acceptable because they contain the BASIC keyword AND. You would not have known that immediately, however. It would have shown up as a syntactical error (?SN) when the program was running.

There are actually two uses for variables. One is for the storage of transient data—information which comes and goes. The other is for constant data—data assigned to the variable and not changed. In the case of constant data, the instruction which sets the variable equal to the constant must be executed once, and then the constant data are available until the variable is, for some reason, changed.

There are three types of numeric variables and one type of string (alphabetic) variable, expressed this way:

%	Integer (whole number) variable (two bytes).
!	Single-precision variable (four bytes).
#	Double-precision variable (eight bytes) is used for mathematical operations where the answer derived would be too large or too small to fit within the four bytes allocated to the single-precision number. This is the default classification. Thus A and A# would appear the same to the machine.
$	String variable, with a range of 0–255 characters, is defined between quotation marks. Exceptions will be discussed later.

An integer is the whole number part of a number. In the real number 33.3, the integer portion is 33. If the variable which was to hold 33.3 were defined as an integer, only the 33 portion would remain. Integers are useful when fractional parts are not needed. They can speed up calculations where a large amount of computation is to be done.

Single-precision variables are most useful for calculating most of the work which will be done. Only when it is necessary to express very large or very small numbers, when calculations must be done with micrometer precision, is the double-precision integer required. For example, to figure your accounts receivable, you probably will not need it.

String variables are used to encase alphabetic data or—and this is an important distinction—a numeric value where the numbers are represented in individual bytes. Each character of a string variable takes one byte, or position of memory. Thus there is a difference between the numeric variable 111 and the string variable "111." If you try to calculate with the latter, you'll be told how erroneous your assumption is. You can get at the value, however, by using the VALue instruction. If A$ = "111" then you can create a numeric value by stating the following: A = VAL(A$). Because string variables are surrounded by quotation marks, you cannot set a string variable equal to quotation marks. This will not work: A$ = " " " ". However, code A$ = CHR$(34) + "KEN" + CHR$(34), and when you PRINT A$, the name will be

surrounded by the quotes. This process is called *variable concatenation,* which will be discussed later.

Rules For Variable Names

As stated, variables names can have any size, but only the first two characters are relevant. They must begin with an alphabetic character, but characters thereafter may be letters, numbers, or decimal points. Numeric variables without a trailing # are assumed to be double-precision variables. Other forms of variables must be declared as integer (%), double-precision (#), or string ($). A class of variables may be defined according to the name of the variable. For instance, consider:

```
10 DEFINT A-G
20 DEFSNG H-K
30 DEFDBL L-R
40 DEFSTR S-Z
```

All variables here that begin with the letters A through G are defined as integer variables. Variables beginning with the letters H through K are single-precision variables. Those beginning with the letters L through R are double-precision variables. Those beginning with letters S through Z are string variables. Once it has been defined by the DEF-method (see the summary at end of this chapter), it is no longer necessary to add the type designator to the variable name. The only real restriction to the use of variable names is confined to the use of reserved words (BASIC keywords).

Following are the reserved words which cannot be included in variable names:

AS	FOR	OFF
ATN	FRE	ON
BEEP	GOSUB	OPEN
CALL	GOTO	OUT
CDBL	HIMEM	PEEK
CHR$	IF	POKE
CINT	INKEY$	POS
CLEAR	INP	POWER
CLOAD	INPUT	PRESET
CLOAD?	INPUT#	PRINT
CLOADM	INPUT$	PRINT#

CLOSE	INSTR	PSET
CLS	INT	READ
COM	IPL	REM
CONT	KEY	RESTORE
COS	KILL	RESUME
CSAVE	LCOPY	RIGHT$
CSAVEM	LEFT$	RND
CSNG	LEN	RUN
CRSLIN	LET	RUNM
DATA	LINE	SAVE
DATE$	LIST	SAVEM
DAY$	LLIST	SCREEN
DEFDBL	LOAD	SGN
DEFINT	LOADM	SIN
DEFSNG	LOG	SOUND
DEFSTR	LPOS	SPACE$
DIM	LPRINT	SQR
EDIT	MAXFILES	STOP
ELSE	MAXRAM	STR$
END	MDM	STRING$
EOF	MENU	TAB
ERL	MERGE	TAN
ERR	MID$	THEN
ERROR	MOTOR	TIME$
EXP	NAME	USING
FILES	NEW	VAL
FIX	NEXT	VARPTR

Avoid using these in combination with other words.

A Place For Everything

A variable is a place to put something, to store alphabetic or numeric data. It may be a constant (A$ = "STRING CONSTANT"). It may be a literal (PRINT "HELLO"). It may be a value used by an instruction (FOR N = 1 TO 10). The point is that variables are places to store data.

The names you assign to variables must, of necessity, occupy space. If you make a habit of using lengthy labels, the available memory will be quickly consumed. Thereafter, each time you define or develop variables, they will utilize the size defined. At some point you will have memory restriction, irrespective of the amount of memory purchased. It's called *Parkinson's Law,* and it will eventually force you to condense your variable names. Using alphabetic letters alone permits the use of 702 variable names (26 one-character names plus 676 two-character names).

BASIC reserves part of memory for use by strings. If no other reservation is made, 256 bytes are reserved. How much memory is available for string storage will depend on the number of other variables and arrays in the program. The system will let you know when you've used up your quota. When it does, you must reserve additional storage for data by using a CLEAR command within the program you're running. CLEAR frees all memory used for data without erasing the program itself in memory and closes any open files. After CLEAR, arrays will be no longer defined, numeric variables will contain a value of zero, string variables will contain a null value, and any DEFinitions established will be lost. The necessary amount of data to be added can be determined only by counting. General practice reveals that a fixed increment, such as 1,000 bytes, is customarily added and is reduced only if that is necessary to contain the program—recall our discussion of MAXRAM and FRE(0). The format of the CLEAR instruction is:

```
CLEAR stringspace, high memory
```

where stringspace is any number (1000 is suggested if the defaulted 256 is insufficient) and high memory is the address above which BASIC is not to be loaded. The high memory option is only necessary when using machine language programs, and is outside the scope of this book.

If there is insufficient space for new variables, BASIC will reuse the allocation of memory and will take time to clean out unused variables (the process is commonly called "garbage collection") for release. You can accelerate this release with the FRE command. However, when you have run out of memory, you have run out of memory. A CLEAR statement will find more memory by clearing out all variables currently in use.

It's important to recognize that there are overheads in your memory allocation. Just because you have purchased the maximum memory configuration doesn't mean that it's all available. Load BASIC and then PRINT FRE(0), and you will find that much of your machine's memory is not available to your BASIC program, string storage is taken from that.

Up to this point we have dealt exclusively with storage space taken by numeric and alphabetic constants. Other places data may reside are:

1. They may reside as a DATA statement (stored within the program itself), which is accessed by a READ instruction. The DATA statement is a permanent constant that can store both alphabetic and numeric data. Once set, these are not modifiable by the BASIC program using them. As with every rule, there is an exception. BASIC DATA statements can be modified by a person knowledgeable machine language.
2. They may reside as an element of an array (arrays will be discussed later). They are used as the place to store data arriving from input media or to develop data targeted for output media.
3. They may reside in files in RAM or cassette. File data are customarily staged in arrays.

Concatenation

It's possible to add alphabetic fields together; one method is called *appending*. Another is called *concatenation*. Note the definitions of the variables A$ and B$ in statements 10 and 20 here. Statement 30 *appends* the two variables to itself into a permanent variable, C$. That combination is displayed in statement 50. In statement 40, however, the two variables are joined temporarily—concatenated—for purposes of the display. Both ways work, but concatenation occupies less memory.

```
10 A$ = "BELL"
20 B$ = "WEATHER"
30 C$ = A$ + B$
40 PRINT A$ + B$
50 PRINT C$
```

The DATA Statement

Consider the following:

```
1000 DATA 42377,BEN,JONES,002376345,9,3,36
1010 DATA 27365,ALICE,JONES,024372122,10,19,36
1020 DATA 34582,KALLY,JONES,073327344,12,12,59
1030 DATA 57263,TERRY,JONES,008328814,9,11,61
1040 DATA 23649,SCOTTY,JONES,012663429,12,4,65
```

Here is structured data about five people, data that will be used by a program to perform some function, perhaps for display in a report. We could call this a personnel file. From left to right, each DATA statement contains a number (perhaps an employee number), a first name, a last name, a social security number, and the month, day, and year of birth. Note that each line is structured in precisely the same manner. Note also that the first item is numeric, the next two are alphabetic, and the remaining four are numeric, though the first of the four, Social Security Number, is treated as alphabetic. Thus, in sequence, they can be described as:

Employee Number (EN)
First Name (FN$)
Last Name (LN$)
Social Security Number (SSN$)
Month of Birth (MOB and MOB$)
Day of Birth (DOB and DOB$)
Year of Birth (YOB and YOB$)

Following is a routine to READ the DATA lines and to present them on the screen and printer. Each is presented twice for comparison of the treatment of the birthdate as numeric or alphabetic. The social security number has been defined as a string variable to preclude suppression of leading zeroes and the presentation of the numbers in mathematical notation.

```
10 CLS
20 LPRINT "Date Of Birth (Numeric)":LPRINT
30 FOR N = 1 TO 5
40     READ EN,FN$,LN$
50     READ SSN$
60     READ MOB,DOB,YOB
70     PRINT EN;"  ";FN$;"  ";LN$
80     PRINT SSN$;"  ";
90     PRINT MOB;"/";DOB;"/";YOB
100    LPRINT EN;"  ";FN$;"  ";LN$;
110    LPRINT SSN$;"  ";
120    LPRINT MOB;"/";DOB;"/";YOB
130 NEXT N
140 PRINT:PRINT:LPRINT:LPRINT
150 RESTORE
160 LPRINT "Date Of Birth (Alphabetic)":LPRINT
170 FOR N = 1 TO 5
180     READ EN,FN$,LN$
190     READ SSN$
200     READ MOB$,DOB$,YOB$
210     PRINT EN;"  ";FN$;"  ";LN$
220     PRINT SSN$;"  ";
```

```
230     PRINT MOB$;"/";DOB$;"/";YOB$
240     LPRINT EN,FN$,LN$,;
250     LPRINT SSN$,;
260     LPRINT MOB$;"/";DOB$;"/";YOB$
270 NEXT N
280 LPRINT:LPRINT:LPRINT
290 END
1000 DATA 42377,BEN,JONES,002376345,9,3,36
1010 DATA 27365,ALICE,JONES,024372122,10,19,36
1020 DATA 34582,KALLY,JONES,073327344,12,12,59
1030 DATA 57263,TERRY,JONES,008328814,9,11,61
1040 DATA 23649,SCOTTY,JONES,012663429,12,4,65
```

Because of the limited nature of variable names, it is generally wise to create a variable table, such as that shown in Figure 6-1. As you examine this program, some questions are certain to arise. What's a RESTORE? Why are the PRINT statements broken down? Isn't this the kind of data that would normally reside on some sort of media?

The pattern of the READ instruction is that DATA line elements are read one at a time in the sequence in which they are called. However, you'd better be very certain that you call for what you need. READs can be single or multiple-variable types, with no punctuation specifications. The PRINT and LPRINT statements are broken down simply because the limited page size of this book doesn't permit full-width instructions or data. Recall that so long as linefeeds are suppressed (by the semicolon), the printing will be contiguous.

Finally, of course, this sort of data customarily resides on magnetic media, RAM, or cassette files. Information ordinarily kept in DATA lines is the sort that is basically unchanging, such as rate tables.

```
*****************************************
*   EN    = EMPLOYEE NUMBER               *
*   FN$   = FIRST NAME                    *
*   LN$   = LAST NAME                     *
*   MOB   = MONTH OF BIRTH (numeric)      *
*   DOB   = DAY OF BIRTH (numeric)        *
*   YOB   = YEAR OF BIRTH (numeric)       *
*   MOB$  = MONTH OF BIRTH (alphabetic)   *
*   DOB$  = DAY OF BIRTH (alphabetic)     *
*   YOB$  = YEAR OF BIRTH (alphabetic)    *
*****************************************
```

Figure 6-1

Sentinel Line

We know that precisely five DATA lines were required for the previous program. And we know (because we program it) what the count is at any time. Suppose, however, that the table is constantly changing and that we must make additions to and deletions from the table. Every change in the table will require a corresponding change in the count of the FOR. . .NEXT loop used to control the READ instructions. That's too much like work.

To get around that hassle, we can add a sentinel line as the last DATA line in the given sequence (DATA lines can be located anywhere within a program and are treated as if they were contiguous). RESTORE resets the pointer to the first DATA record. As a matter of convenience, however, DATA lines are customarily located en masse at the end of the program. This sentinel line is something uniquely distinguishable and unlikely to occur naturally. The most commonly used sentinels are 9s for numeric fields and Zs for alphabetic fields. As a result, the loop (FOR. . .NEXT) must be structured toward an impossible-to-reach number, and a test for the sentinel must be contained within the FOR. . .NEXT loop, as follows:

```
10 CLS
20 FOR N = 1 TO 1000
30     READ EN,FN$,LN$
40     READ SSN$
50     READ MOB$,DOB$,YOB$
60     IF EN = 99999 THEN 160
70     PRINT EN;"  ";FN$;"  ";LN$
80     PRINT SSN$;"  ";
90     PRINT MOB$;"/";DOB$;"/";YOB$
100    LPRINT EN;"  ";FN$;"  ";LN$;
110    LPRINT SSN$;"  ";
120    LPRINT MOB$;"/";DOB$;"/";YOB$
130 NEXT N
140 PRINT:PRINT:LPRINT:LPRINT
150 END
160   PRINT:PRINT "SENTINEL LOCATED"
170   LPRINT:LPRINT "SENTINEL LOCATED"
180 PRINT:PRINT:LPRINT:LPRINT
190 END
1000 DATA 42377,BEN,JONES,002376345,9,3,36
1010 DATA 27365,ALICE,JONES,024372122,10,19,36
1020 DATA 34582,KALLY,JONES,073327344,12,12,59
1030 DATA 57263,TERRY,JONES,008328814,9,11,61
1040 DATA 23649,SCOTTY,JONES,012663429,12,4,65
```

The following shows that the sentinel was indeed located:

```
42377       BEN         JONES       002376345    9/3/36
27365       ALICE       JONES       024372122    10/19/36
34582       KALLY       JONES       073327344    12/12/59
57263       TERRY       JONES       008328814    9/11/61
23649       SCOTTY      JONES       012663429    12/4/65

SENTINEL LOCATED
```

Array

Although the DATA statement is a good way to provide temporary input to a program, it is not the only way to introduce data to the system. Another is the use of an array. As previously stated, data retrieved from a diskette is placed into the array.

An array can be thought of as a series of mailboxes or pigeon holes into which items of data can be sorted. The two types of arrays are (1) unidimensional and (2) multidimensional.

The unidimensional array is a sequential list, using separate variables. Figure 6-2 is an example of a six-element array (A$) with locations (mailboxes) 0 through 5 (yes, 0 is a number):

```
.................
.    A$(0)      .
.................
.    A$(1)      .
.................
.    A$(2)      .
.................
.    A$(3)      .
.................
.    A$(4)      .
.................
.    A$(5)      .
.................
```

Figure 6-2

The multidimensional array is comprised of rows and columns, with boxes across the horizontal span, and boxes along the vertical span. Figure 6-3 is an example of an array consisting of four rows of four elements (mailboxes):

```
.....................................
. B(0,0) . B(0,1) . B(0,2) . B(0,3) .
.....................................
. B(1,0) . B(1,1) . B(1,2) . B(1,3) .
.....................................
. B(2,0) . B(2,1) . B(2,2) . B(2,3) .
.....................................
. B(3,0) . B(3,1) . B(3,2) . B(3,3) .
.....................................
```

Figure 6-3

DIM

To use an array, you must specify its characteristics to BASIC—the process is called *DIMensioning*. DIM requires an instruction within the program after any CLEAR statement. The DIM statements for the previously illustrated arrays look like this:

```
10 DIM A$(5)
20 DIM B(2,3)
```

If you're sharp, something won't look quite right. The illustration of A$ had *six* elements, not five. The illustration of Array B had 12 mailboxes, not the six you asked for. There is, however, a common thread to the quandry. Of the six boxes, one of them had a zero reference. Of the 12 boxes, six had a zero reference. Therefore, the DIMension statement identifies the *top* position relative to zero. You can use the zero-referenced places; in practice, you'll use them for work space.

Go back to the multidimensional array and darken the outline of boxes B(1,1), B(1,2), B(1,3), B(2,1), and B(2,3). For all intents and purposes, these are the working elements of the array. The first position to be referenced, therefore, is B(0,0) plus one. The "plus one" is commonly called a *subscript*. To illustrate it, consider the following:

```
10 DIM A$(5)
20 CLS
30 FOR N = 1 TO 5
40     A$(N) = "ELEMENT " + STR$(N)
50 NEXT N
60 FOR N = 1 TO 5
70     PRINT A$(N)
80 NEXT N
90 END
```

In this program, we have dimensioned the single-dimensioned array and have loaded a value into it. The program loads the word "ELEMENT" followed by a number into the array. The new instruction in statement 40—STR$—instructs the computer to store the string value of the number (in this case, the position within the array) in the array itself. Because of it, statement 40 concatenates and stores two values in the array. Since it is impossible to concatenate an alphabetic value and a numeric value and since the FOR. . .NEXT loop is controlled by the numeric variable, it is necessary to convert the numeric value to an alphabetic value. STR$ is the precise opposite of the VAL function (see the summary at the end of this chapter).

Here's a similar program to load a multidimensioned array, using C as the constant variable:

```
10 DIM B(2,3)
20 C = 111
30 CLS
40 FOR N = 1 TO 2
50     FOR X = 1 TO 3
60         B(N,X) = C
70         C = C + 111
80     NEXT X
90 NEXT N
100 FOR N = 1 TO 2
110     FOR X = 1 TO 3
120         LPRINT "ELEMENT B(";N;",";X;") = " B(N,X)
130     NEXT X
140 NEXT N
150 FOR N = 1 TO 3:LPRINT:NEXT N
160 END
```

Enter and run the program and you'll see that the several elements of the array have indeed been filled—with an incremental constant. In case you're wondering about the spacing of the printed output (between the parentheses, as shown), recall that all numbers are signed and the positive sign (+) does not print. To avoid the spacing you would have to complicate the PRINT instruction to insert a backspace, ASCII 008.

```
ELEMENT B( 1 , 1 ) =  111
ELEMENT B( 1 , 2 ) =  222
ELEMENT B( 1 , 3 ) =  333
ELEMENT B( 2 , 1 ) =  444
ELEMENT B( 2 , 2 ) =  555
ELEMENT B( 2 , 3 ) =  666
```

There are, of course, similarities between the loading of arrays from DATA lines and the loading of arrays from magnetic diskette files. So far we've concentrated on the mechanics of loading arrays, with no concept of what the array represented. Let us now use the personnel file illustration presented previously.

Examine the DATA lines and you will see there are six rows of seven elements. Therefore, the DIMension must be 6 by 7. Let's call the array by its name—PF$ (for personnel file). An array must be either numeric or alphabetic, but never both or mixed. Since we have a mixture of data types, let's treat them all as alphabetic, because we know that with the STR$ and VAL functions we can change them. Following is the first record in detail:

```
PF$(1,1)  =  42377
PF$(1,2)  =  BEN
PF$(1,3)  =  JONES
PF$(1,4)  =  002376345
```

```
PF$(1,5)  =  9
PF$(1,6)  =  3
PF$(1,7)  =  36
```

Alice's record would then begin at 2,1; Kally's at 3,1; etc.

Before we proceed, let's load the DATA lines into the array and print them to ensure that we understand what is where:

```
10 CLS
20 FOR N = 1 TO 1000
30     PRINT "READING RECORD: ";N
40     FOR X = 1 TO 7
50         READ PF$(N,X)
60         PRINT "  ";PF$(N,X);
70         IF PF$(N,X) = "99999" THEN 120
80     NEXT X
90     PRINT
100 NEXT N
110 END
120 PRINT:PRINT "SENTINEL LOCATED"
130 END
1000 DATA 42377,BEN,JONES,002376345,9,3,36
1010 DATA 27365,ALICE,JONES,024372122,10,19,36
1020 DATA 34582,KALLY,JONES,073327344,12,12,59
1030 DATA 57263,TERRY,JONES,008328814,9,11,61
1040 DATA 23649,SCOTTY,JONES,012663429,12,4,65
1050 DATA 99999,ZZZZZ,ZZZZZ,ZZZZZ,ZZZZZ,ZZZZZ,ZZZZZ
```

As you examine this program note that you did not have to identify the original names assigned and then transfer them to the array. It was necessary only to identify the array parts. We used variable X to scan the columns and variable N to scan the rows, producing this output:

```
READING RECORD:  1
  42377  BEN  JONES  002376345  9  3  36
READING RECORD:  2
  27365  ALICE  JONES  024372122  10  19  36
READING RECORD:  3
  34582  KALLY  JONES  073327344  12  12  59
READING RECORD:  4
  57263  TERRY  JONES  008328814  9  11  61
READING RECORD:  5
  23649  SCOTTY JONES  012663429  12  4  65
READING RECORD:  6
  99999
SENTINEL LOCATED
```

Look carefully at the previous program. Note that it does not contain a DIMension statement. Why? The reason is that BASIC will automatically define a minimum 10-element matrix for you without this

statement. For any larger matrix, however, BASIC will give you an error message (?BS ERROR) requiring one.

Strings and Things

In business applications you will find yourself manipulating more nonnumeric data than you will numeric data. The specification of "nonnumeric" should not be arbitrarily assumed to be alphabetic, for as we have seen, it's possible to treat numeric data (from DATA lines or from diskette) as if they were alphabetic (nonnumeric) data. Therefore we must learn how the MEW handles string data.

The first three instructions required can be used to "break down" an alphabetic string. They are LEFT$, MID$, and RIGHT$. With LEFT$ and RIGHT$, you can specify the number of characters you wish to extract (copy) from the left or right side of the string. With MID$ you can extract from the center (or anywhere along the entire length) of the string; the only difference here is that you must specify the beginning position of the extraction. In the example given, LEFT$, RIGHT$, and MID$ are the *source* of data. As the *target* of data, only MID$ is workable.

Enter the following program and run it, and you will achieve the results shown in the output.

```
10 CLS
20 C3$ = " THREE CHARACTERS: "
30 TEST$ = "ABCDEFGHI"
40 PRINT " POSITION:  123456789"
50 PRINT "CHARACTER: ";TEST$
60 PRINT
70 PRINT "  LEFT" + C3$;LEFT$(TEST$,3)
80 PRINT "MIDDLE" + C3$;MID$(TEST$,4,3)
90 PRINT " RIGHT" + C3$;RIGHT$(TEST$,3)
100 GOTO 100
```

That produces this output:

```
 POSITION:  123456789
CHARACTER:  ABCDEFGHI

  LEFT THREE CHARACTERS:  ABC
MIDDLE THREE CHARACTERS:  DEF
 RIGHT THREE CHARACTERS:  GHI
```

All this is well and good, and we have extracted from a string whose size is known to be a specified number of characters. Suppose,

however, that we were searching through data for a specified combination of characters, such as might be used in a keyword. What then? The INSTR function will tell where in the string the specific sequence of characters is located. For instance, if we wished to locate the substring EFG from our brief alphabetic sequence, we would assume that the computer would tell us that it is located in the fifth position. Once known, the data can be accessed. Enter and try the following program:

```
10 CLS
20 TEST$ = "ABCDEFGHI"
30 INPUT "ENTER THE STRING YOU SEEK: ";A$
40 X = INSTR(TEST$,A$)
50 IF X > 0 THEN PRINT "THE LOCATION OF ";A$;" IS";X:GOTO 70
60 PRINT A$;" IS NOT LOCATED IN THE TARGET STRING"
70 PRINT
80 GOTO 20
```

These messages will bend the line on the display, but it demonstrates the point.

Occasionally the length of a given string of data is important to know. Let's say that you can print only the first 10 characters of a field on a page of printed output. To determine the length, the LENgth function is used. Once the length is known, you can determine the size of the data to be presented. For instance, consider the following:

```
10 CLS
20 TENS$ = "         1         2         3         4"
30 UNIT$ = "12345678901234567890123456789012345678901234567890"
40 PRINT "SCALE:"
50 PRINT TEN$;
60 PRINT UNIT$;
70 INPUT "ENTER STRING: ";A$
80 INPUT "SIZE OF OUTPUT: ";SIZE
90 PRINT "RESULTING STRING: ";
100 IF LEN(A$) < SIZE THEN PRINT "IMPOSSIBLE!":GOTO 120
110 PRINT LEFT$(A$,SIZE)
120 PRINT "PRESS ANY KEY TO CONTINUE"
130 Z$ = INKEY$:IF Z$ = "" THEN 130 ELSE 10
```

Here a variable-length field is entered, and we have the opportunity to specify the output length. This approach would work as surely as if we were to trim all variables down to a common size. First, observe the semicolons at the end of the PRINT statements at 50 and 60. Because these constants are precisely 40 characters in width, the linefeed must be suppressed, else the printing would be separated by a line. Observe also the ELSE command in statement 130. The ELSE is a default which functions as if it were a GOTO in the following statement. In this particular example, the coding advises you if it is impossible to specify

a length for a field to be entered. In actuality, you would accept any length up to and including the length specified. The output should look like this:

```
                        1         2         3         4
SCALE:          12345678901234567890123456789012345678901

ENTER STRING:   ABCDEFGHIJKLMNOPQRSTUVWXYZ

SIZE OF OUTPUT: 15

RESULTING STRING: ABCDEFGHIJKLMNO

PRESS ANY KEY TO CONTINUE
```

The last two string functions to be discussed are SPACE$ and STRING$. The former will allow you to insert a specified number of spaces. Thus SPACE$(5) would put five spaces wherever you need them. STRING$ allows you to move a prescribed number of ASCII or string characters. PRINT STRING$(80,223), for instance, will light up one line of the screen. Try it. It will have some value when we try to use the line graphics on the display. Both these instructions will be discussed in greater detail when we discuss graphics.

Binary Search

One of the better techniques commonly used in computer applications is the *binary search*. Binary means *two* and the principle of the binary search assumes:

1. That whatever you're searching for is in some predetermined sequence, usually ascending.
2. That the field of search can be narrowed merely by using the "rule of halves," that is, by dividing the object of the search into equal parts, determining the relationship of the item for which you're searching to the midpoint, selecting the proper half, dividing *it* into two equal parts, and then continuing in the same fashion until the required item is found. In other words, a computerized twenty-questions game.

Suppose that there is a very long list, such as a list of names, perhaps of a thousand names. If the list were compiled in any random sequence, it would be very difficult to find the name of any one person, say John Smith. What is needed is a list arranged in alphabetical sequence, beginning with A and ascending through the alphabet to Z. Now, when you wish to search the list for Smith, your eyes fall to the

Ss, you look at them one by one until you find the Smiths, and then, you look for John. The computer works in the same way, except it cannot "see" the Ss.

After the list has been put into alphabetic sequence, it must be divided down the middle, between M and N. The name Smith (S) is checked against N and found to be larger since it is known that S falls somewhere between N and Z. The "bottom half" of the alphabet (A to M) is therefore not searched. The next step is to split the upper half of the alphabet so that the midpoint is now T. Since S is smaller than T, the upper quarter of the alphabet is no longer searched. It is known that S falls somewhere between N and T, etc.

This can be illustrated by a game, generally called "guess the number"; it can be programmed as follows:

```
10 CLS
20 DEFINT A,C,R
30 PRINT "PICK A NUMBER FROM 1 TO 100":PRINT
40 PRINT "PRESS ANY KEY TO CONTINUE"
50 Z$ = INKEY$
60 IF Z$ = "" THEN 50
```

These instructions merely give you time to select a number in the range of 1 to 100. You do not need to tell that number to the computer. Simply write it down or keep it in mind. The integer definitions of statement 20 will ensure we are dealing with whole numbers. Numeric variable A is used as part of an ON. . .GOTO statement; C is used to keep track of your guesses. The instructions in statements 50 and 60 form a tight loop that waits for the keyboard entry.

The computer doesn't know the number you have in mind, of course, but it does know the range it has offered you and the extremes and midpoint of that range because of the additional following statements:

```
70 RM = 50:RT = 100:RB = 0
80 CLS
90 GOSUB 210

                                  ************************
                                  * RM = Range, midpoint *
                                  * RT = Range, top      *
                                  * RB = Range, bottom   *
                                  ************************
```

These statements define the midpoint, clear the screen, and establish the linkage to a subroutine. In the subroutine, the following alternatives will be displayed:

```
210 '************************************
220 '*       DISPLAY SUBROUTINE         *
230 '************************************
240 CLS
250 PRINT "IS IT:"
260 PRINT TAB(5);"1.  GREATER THAN ";RM
270 PRINT TAB(5);"2.  EQUAL TO ";RM
280 PRINT TAB(5);"3.  LESS THAN ";RM
290 PRINT
300 PRINT "WHICH";
310 Z$ = INKEY$
320 IF Z$ = "" THEN 310
330 IF VAL(Z$) > 3 OR VAL(Z$) < 1 THEN 310
340 A = VAL(Z$)
350 C = C + 1
360 RETURN
```

The comments are outlined here in asterisks. If you have room, use some of them for clarity. However, be aware that they may occupy memory you will need for your program, and use them accordingly.

Here we have a simple menu arrangement with three options. Presentation of the menu is straightforward, right down to statement 330. What happens there? Recall that INKEY$ accepts an alphabetic input. Because we wish to use numeric variable A for an ON. . .GOTO, the input value must be changed to numeric. Simultaneously, the validity of the entry is error-trapped, and if you enter a value greater than 3, it will be rejected. The same is true if you try a value less than 1 (0, − 1, etc.). Try it. The integer variable A is established in statement 340, the "guess" counter is incremented in 350, and the computer leaves the subroutine at 360.

In the subroutine, you are asked to compare the number in your head to the current midpoint. "Greater than" is option 1; "equal to" is option 2; and "less than" is option 3. These values correspond to the ON. . .GOTO in statement 100, shown with four new statements below. In this program, only the "equal to" has been coded, because at some point you must respond to this condition (if you're honest).

```
100 ON A GOTO ___, 120, ___
110 GOTO 70
120 PRINT "NUMBER GUESSED IN ";
130 PRINT C;" TRIES"
140 END
```

Statement 100 is amplified here, along with four new statements:

```
100 ON A GOTO 150, 120, ___

150 RB = RM
160 GOSUB 370
170 GOTO 80
```

Now we can concentrate upon the "greater than" side of the option. In the "greater than" situation, the range is 50 to 100. The first thing to do, therefore, is to establish that range. Recall that the program is being developed to make the routines useful no matter what the numbers may be. We now know, however, that there is a new midpoint (RM). Before the old midpoint value is destroyed, the variable used for the bottom of the range (RB) must be established with the current value of variable RM. That occurs in statement 150. Now a new midpoint must be established (the subroutine at 370), although the upper limit (RT) does not need adjustment. Here is that subroutine:

```
370 '************************************
380 '*      UPPER HALF SUBROUTINE       *
390 '************************************
400 RM = (RM + RT)/2
410 RETURN
```

This sends us back again to statement 100 and an additional set of statements:

```
100 ON A GOTO 150, 120, 180

180 RT = RM
190 GOSUB 420
200 GOTO 80
420 '************************************
430 '*      LOWER HALF SUBROUTINE       *
440 '************************************
450 RM = (RM + RB)/2
460 RETURN
```

On the "less than" side, the range is now 0 to 50, although those values cannot be used directly. The upper limit must be adjusted, which is done in statement 180. The lower limit remains as it is. It is now necessary to find a new midpoint, a job done by the subroutine at 420.

If the variables had not been defined as integers, there would have been one minor problem. When the ranges are such that the midpoint develops a decimal (for example, half of 25 is 12.5), a specific position cannot be determined, and a number cannot be selected from the list. To ensure that this does not happen (if the variable has not been defined as an integer), a new instruction is required—the INTeger instruction. The INTeger instruction selects only the whole number portion of a number, disregarding the decimal fraction. It may not be absolutely precise, but it suits the problem. The general format of the instruction is A = INT(B). In this particular program, you will become aware of the problem if you continue to answer the menu on the high side. After it reaches 99, it will go to 100 and stay there. The same is true on the low side, where it rests at zero.

Here is the program in its entirety:

```
10 CLS
20 DEFINT A,C,R
30 PRINT "PICK A NUMBER FROM 1 TO 100":PRINT
40 PRINT "PRESS ANY KEY TO CONTINUE"
50 Z$ = INKEY$
60 IF Z$ = "" THEN 50
70 RM = 50:RT = 100:RB = 0
80 CLS
90 GOSUB 210
100 ON A GOTO 150, 120, 180
110 GOTO 70
120 PRINT "NUMBER GUESSED IN ";
130 PRINT C;" TRIES"
140 END
150 RB = RM
160 GOSUB 370
170 GOTO 80
180 RT = RM
190 GOSUB 420
200 GOTO 80
210 '************************************
220 '*       DISPLAy sUBROUTINE         *
230 '************************************
240 CLS
250 PRINT "IS IT:"
260 PRINT TAB(5);"1.  GREATER THAN ";RM
270 PRINT TAB(5);"2.  EQUAL TO ";RM
280 PRINT TAB(5);"3.  LESS THAN ";RM
290 PRINT
300 PRINT "WHICH";
310 Z$ = INKEY$
320 IF Z$ = "" THEN 310
330 IF VAL(Z$) > 3 OR VAL(Z$) < 1 THEN 310
340 A = VAL(Z$)
350 C = C + 1
360 RETURN
370 '************************************
380 '*      UPPER HALF SUBROUTINE       *
390 '************************************
400 RM = (RM + RT)/2
410 RETURN
420 '************************************
430 '*      LOWER HALF SUBROUTINE       *
440 '************************************
450 RM = (RM + RB)/2
460 RETURN
```

Spend some time tracing the program through using your own data. You'll find that it works not only for the upper half and the lower half, but also for the upper quarter of the lower half and the lower quarter of the upper half. Once that is accomplished, finding the number that you dreamed up is not difficult; in fact, you should find it within 18 tries. More sophisticated routines for "guessing the number" exist, but

this example illustrates the binary search principle. You might have been able to figure from 1 to 100, but suppose the range had been from 10,000 to 10,000,000. It might have been a little more difficult.

Searching the Table

Taking the process a step further, this technique might be applied to searching a table for the presence of a number. Up to this point, an absolute match has been the goal. To search a number table, remember that the number may or may not be in the table and that it is our responsibility to search for it.

The first step, naturally, is to build a table. It is possible to take one from a data file (diskette or DATA lines), but even that would have to be built first. To illustrate the principle, we'll do a little simple construction using random numbers. The Model 100 can generate numbers in the range of zero to one. The numbers look like these, drawn from a "power up" generation:

```
.59521943994623
.10658628050158
```

The table which is generated is fixed in nature, that is, the same two numbers would appear on each power up; the list would continue for some extended time. Naturally, these are decimal, not integer values, so for them to be useful, they will have to be scaled and made to be integers.

We'll generate our own table. Assume that the table consists of three elements—a stock number (in the range of 1 to 10,000), a description (which will be generated merely for illustrative purposes), and a quantity-on-hand (in the range of 1 to 10,000). Enter and run the following program:

```
10 CLS
20 LPRINT
30 FOR X = 1 TO 20
40     FOR Z = 1 TO 5
50         LPRINT INT(RND(X) * 10000),
60     NEXT Z
70 LPRINT
80 NEXT X
90 MENU
```

In this example, the program generates 100 random numbers and prints them in zones (trailing comma on statement 50) in groups of five (FOR. . .NEXT loop of statements 40 and 60). At the end, rather than

specifying END and returning to BASIC, the instruction MENU is specified, returning you to the main menu.

The RND instruction of statement 50 generates a decimal value less than one; multiplying it by a large value places it into the integer (greater than 0) range. What is actually generated is the integer of a random number (RND(X)). The 10,000 multiplier places the number into the 1 to 10,000 range. This was generated by my Model 100; yours may be different:

```
5952    1065    7659    5775    7347
1842    3707    9495    6379    4704
8321    4979    1858    1295    4905
7725    281     4877    6536    4503
8255    8153    5816    368     2539
3686    1775    2214    7720    1338
2666    4367    4350    4886    9600
6899    1248    1482    258     791
1331    3481    8133    5661    9410
3594    2163    9914    4834    796
5056    6469    9460    1874    8620
6891    7348    2249    7046    8273
1834    2464    6005    2508    4577
6643    5952    2962    9969    7773
1004    7793    2579    9792    675
1278    8408    512     4709    2115
5414    7023    895     4641    9722
1635    9392    3578    8982    5026
1577    5996    7323    4091    5294
6806    8783    9299    3617    5783
```

You'll note that since these numbers do not appear in any specific sequence, it will be difficult to do a binary search. To be usable, they must be arranged within an ascending sequence. Before we do that, however, we need to develop an array, or inventory table (DIM INVTAB$(100,3)) into which we can store the item number, the description, and the quantity-on-hand. Because no calculation is done on these figures, the table is a string variable table, which means that the number generated by the table program must be changed somewhat. The following is sufficient to generate and display the array and for us to build upon:

```
10 CLS
20 CLEAR 1000
30 DIM INVTAB$(100,3)
40 '************************************************
50 '*      THIS ROUTINE GENERATES THE TABLE         *
60 '************************************************
70 FOR X = 1 TO 10
80     Y = X + 2
90     STKNUM = INT(RND(Y) * 10000):IF STKNUM <1000 THEN 90
```

```
100     INVTAB$(X,1) = STR$(STKNUM)
110     INVTAB$(X,2) = "ITEM DESCRIPTION"
120     Z = X + 5
130     QOH = INT(RND(Z) * 10000)
140     INVTAB$(X,3) = STR$(QOH)
150 NEXT X
160 '************************************************
170 '*   THIS ROUTINE DISPLAYS THE GENERATED TABLE   *
180 '************************************************
190 FOR X = 1 TO 10
200     PRINT INVTAB$(X,1);TAB(20);
210     PRINT INVTAB$(X,2);TAB(50);
220     PRINT INVTAB$(X,3)
230 NEXT X
```

At this point, interesting things begin to happen. Recall that the premise of a binary search is that a table may or may not contain a stock number. As the table is presently constructed, such a search is impossible, since the table must be in ascending order on the key field. The key field is that field where sequence will be accomplished. In this case, it's the first of the three array elements—item number. Consequently, the table must be sorted.

Of the many ways to perform a sort, the easiest to construct and explain is called the *exchange sort*. The exchange sort is useful for small amounts of data, but should not be used for extensive files. Another type of sort, more useful for that purpose, is the *Shell-Metzner* Sort, which is explained in the Model 100's Reference Book.

The exchange sort examines array elements in pairs, changing the positions of each pair within the table so that the largest constituent of the pair is in the higher position within the array. In this manner, array element 1 (corresponding to array elements 1, 2, and 3, which contain the complete record) is compared with array element 4 (corresponding to array elements 4, 5, and 6—the next record). In a multidimensioned array, we're really talking about (1,Y) being compared to (2,Y), etc. If the second record is larger than the first record (a higher inventory number), or if it is equal, then nothing changes except that you are now able to compare the second and third pairs, and the process continues. In this instance, we won't worry about eliminating duplicate table entries. If, on the other hand, the first record is larger than the second, the first record must be removed from the table and placed in a temporary work area, the second record must take over the space formerly occupied by the first record, and the first record must then be removed from the temporary work area and take over the space formerly occupied by the second record.

When you're sorting and an exchange is made, you must identify that the exchange has, in fact, occurred. Two numbers can be exchanged that are not only out of sequence between themselves but also

out of sequence within the entire set of numbers. Thus we set an internal switch (often called a *flag*). The switch is set to "ON" or 1 when an exchange has been made. The program can then be told to continue through the list of data until the flag has not been set, i.e., no exchange has been made. When the program goes entirely through a table without making an exchange, then the table is in sequence. This procedure requires additional program statements, as follows:

```
240 '************************************************
250 '* TERMINATE THE ARRAY WITH A SENTINEL; SET FLAG *
260 '************************************************
270 FOR Y = 1 TO 3
280     INVTAB$(X,Y) = "ZZZZZ"
290 NEXT Y
300 FLAG$ = "OFF"
310 '************************************************
320 '*        EXCHANGE SORT THE ARRAY ROUTINE        *
330 '************************************************
340 FOR X = 1 TO 100
350     IF INVTAB$(X,1) = "ZZZZZ" THEN 480
360     IF INVTAB$(X + 1,1) > = INVTAB$(X,1) THEN 470
370     ITEMTEMP$ = INVTAB$(X,1)
380     DESCTEMP$ = INVTAB$(X,2)
390     QTYTEMP$ = INVTAB$(X,3)
400     INVTAB$(X,1) = INVTAB$(X + 1,1)
410     INVTAB$(X,2) = INvTAB$(X + 1,2)
420     INVTAB$(X,3) = INVTAB$(X + 1,3)
430     INVTAB$(X + 1,1) = ITEMTEMP$
440     INVTAB$(X + 1,2) = DESCTEMP$
450     INVTAB$(X + 1,3) = QTYTEMP$
460     FLAG$ = "ON"
470 NEXT X
480 IF FLAG$ = "OFF" THEN 540
490 FLAG$ = "OFF"
500 GOTO 340
```

The flag is now set, the table is sealed with sentinels, and the sorting process begins. We now add the following routine:

```
510 '************************************************
520 '*     THIS ROUTINE DISPLAYS THE SORTED TABLE    *
530 '************************************************
540 PRINT
545 FOR X = 1 TO 10
550     PRINT INVTAB$(X,1);TAB(20);
560     PRINT INVTAB$(X,2);TAB(50);
570     PRINT INVTAB$(X,3)
580 NEXT X
590 END
```

Run it and you'll note something peculiar. A stock number may be smaller than 1000 but nevertheless is left justified in the number field.

That's because we have treated this field as an alphabetic field, and alphabetic fields align to the *high order* (left justified), whereas numeric fields align to the *low order* (right justified). This problem can be solved several ways, but since this is a demonstration program only, change statement 90 as follows:

```
90      STKNUM = INT(RND(X) * 10000):IF STKNUM <1000 THEN 90
```

Here is the program in its entirety:

```
10 CLS
20 CLEAR 1000
30 DIM INVTAB$(100,3)
40 '*********************************************
50 '*      THIS ROUTINE GENERATES THE TABLE      *
60 '*********************************************
70 FOR X = 1 TO 10
80     Y = X + 2
90     STKNUM = INT(RND(Y) * 10000):IF STKNUM <1000 THEN 90
100    INVTAB$(X,1) = STR$(STKNUM)
110    INVTAB$(X.2) = "ITEM DESCRIPTION"
120    Z = X + 5
130    QOH = INT(RND(Z) * 10000)
140    INVTAB$(X,3) = STR$(QOH)
150 NEXT X
160 '*********************************************
170 '* THIS ROUTINE DISPLAYS THE GENERATED TABLE  *
180 '*********************************************
190 FOR X = 1 TO 10
200     PRINT INVTAB$(X,1);TAB(20);
210     PRINT INVTAB$(X,2);TAB(50);
220     PRINT INVTAB$(X,3)
230 NEXT X
240 '************************************************
250 '* TERMINATE THE ARRAY WITH A SENTINEL; SET FLAG *
260 '************************************************
270 FOR Y = 1 TO 3
280     INVTAB$(X,Y) = "ZZZZZ"
290 NEXT Y
300 FLAG$ = "OFF"
310 '************************************************
320 '*        EXCHANGE SORT THE ARRAY ROUTINE        *
330 '************************************************
340 FOR X = 1 TO 100
350     IF INVTAB$(X,1) = "ZZZZZ" THEN 480.
360     IF INVTAB$(X + 1,1) > = INVTAB$(X,1) THEN 470
370     ITEMTEMP$ = INVTAB$(X,1)
380     DESCTEMP$ = INVTAB$(X,2)
390     QTYTEMP$ = INVTAB$(X,3)
400     INVTAB$(X,1) = INVTAB$(X + 1,1)
410     INVTAB$(X,2) = INVTAB$(X + 1,2)
420     INVTAB$(X,3) = INVTAB$(X + 1,3)
430     INVTAB$(X + 1,1) = ITEMTEMP$
440     INVTAB$(X + 1,2) = DESCTEMP$
```

```
450      INVTAB$(X + 1,3) = QTYTEMP$
460      FLAG$ = "ON"
470 NEXT X
480 IF FLAG$ = "OFF" THEN 540
490 FLAG$ = "OFF"
500 GOTO 340
510 '*************************************************
520 '*      THIS ROUTINE DISPLAYS THE SORTED TABLE     *
530 '*************************************************
540 PRINT
545 FOR X = 1 TO 10
550      PRINT INVTAB$(X,1);TAB(20);
560      PRINT INVTAB$(X,2);TAB(50);
570      PRINT INVTAB$(X,3)
580 NEXT X
590 END
```

The exchange sort is not the fastest sort, as stated, but it is very effective and easily implemented. This particular example has been structured on the basis of 100 numbers. If the amount to be sorted is sufficiently large, it would be wise to add instructions to display the items as they are being sorted, so you are assured that something is happening. Again, the exchange sort should not be used for large files.

Table Maintenance

The actual input/output with RAM files is not complicated. We've generated data for a table and have sorted it. At this stage, the ability to write a table on a RAM file or cassette or to the printer in the form of a report is simply an interface step.

We will now concentrate on building a table directly from input entered at the keyboard and on the routines needed to add to the table, delete from the table, change a table entry, sort it on any data elements (using the exchange method), and report from it. Let's make a decision: Our table will never exceed 50 items. And we're going to do something unique with our table—we're going to store musical notes. After all, how does a collection of musical notes differ significantly from a warehouse full of widgets?

Let's consider *all* the features that are necessary to table handling:

1. We must be able to add to the table, and do so in any sequence.
2. We must be able to change an entry already made to the table.
3. We must be able to delete a table entry.
4. We need simple inquiry facilities; we might not want to make a second entry under the same control element.

5. We need a facility to display and (upon selection of the option) to produce a hardcopy output of the table.
6. We need a facility to sort the table in preparation for its storage on magnetic media. However, since we know the sequence in which we would like our musical composition, sorting of the table does not apply in this instance, and will therefore be omitted.
7. We need interfaces to load the table from magnetic media and dump it to magnetic media. Therefore, we will explore RAM files, CASsette files, MoDeM (MDM) files, and COMmunication files.

As you might surmise, these requirements could produce a program of considerable size. How much facility will be built into the program will be a function of your available memory. We'll develop the application in pieces which can be combined according to your needs. Those pieces are called *modules*. If memory is limited, these modules must be selected and combined. The maximum MEW memory size should be more than sufficient, however.

The Musical MEW

Before we plunge directly into the music program which looks like an inventory program, let's lay a little ground work and firm up a few concepts. Believe it or not, the following program plays "Yankee Doodle:"

```
10 RESTORE
20 FOR N = 1 TO 32
30     READ A
40     SOUND A,10
50 NEXT N
60 MENU
100 DATA 2348,2348,2092,1864
110 DATA 2348,1864,2092,10
120 DATA 2348,2348,2092,1864
130 DATA 2348,2348,2484,10
140 DATA 2348,2348,2092,1864
150 DATA 1758,1864,2092,2348
160 DATA 2484,3134,2793,2484
170 DATA 2348,10,2348,10
```

First, statement 10, RESTORE, sets the pointer to the first of the DATA lines. Next, the SOUND instruction of statement 40 has two parameters: frequency and duration. The duration has been fixed at 10, which will be explained. The value of the variable A is established by the READ instruction which accesses one element of the DATA table. The FOR. . .NEXT loop is established to permit READing of 32 suc-

cessive values, which is precisely the number of elements in the DATA table.

The SOUND instruction will cause a note to be generated upon the designated frequency for a period of one second for each value of 50. The timing is fixed here at 10 to provide a staccato note at a higher than normal tempo. The frequencies in the DATA line are derived from the following table. The value of 10 in the DATA lines is strictly arbitrary. The number is outside the ability of the machine to SOUND, so in essence we have told the computer not to sound on that frequency for the designated period of time. In this instance the period of time is fixed. In the program which will follow this discussion, you will have the opportunity to vary the timing. Table 6-1 is the frequency table for the five available octaves on the Model 100:

Table 6-1

	Octave				
Note	1	2	3	4	5
G	12538	6269	3134	1567	783
G#	11836	5918	2969	1479	739
A	11172	5586	2793	1396	698
A#	10544	5272	2636	1318	659
B	9962	4976	2484	1244	622
C	9394	4697	2348	1174	587
C#	8866	4433	2216	1108	554
D	8368	4184	2092	1046	523
D#	7900	3950	1975	987	493
E	7456	3728	1864	932	466
F	7032	3516	1758	879	439
F#	6642	3321	1660	830	415

This table has been extracted from the Radio Shack Model 100 manual. The numbers are not precisely the way they are on a musical scale, however. For example, the note A above middle C is at 440 Hz. Possibly the manufacturer sought to establish a range to permit the most

latitude within the capabilities of the machine. Select any row and follow it from left to right and you will see that each successive octave halves the frequency. This is called a *harmonic*. If you are not musically inclined, simply remember that on a violin or guitar, the shorter the string has been made (by pressing a finger on it), the higher the pitch. Thus, as these numbers become smaller, it is analogous to shortening the strings.

When the MEW is turned on, the SOUND capability is enabled. It may be disabled with the command SOUND OFF. And it may be reenabled with the command SOUND ON.

Musical Inventory

The program we are about to develop builds tables, as we have previously discussed. Just as an inventory table would contain the item number, description, and quantity on hand, our table will contain the sequence of the note in the set called music, the frequency of the note to be played, and the duration for which it will be played. As an inventory program would provide a list of inventory, this program will provide a list of the music in three ways: (1) it will cause the computer to play (<S>ING) the music; (2) it will provide a display of the numbers from the table on the screen; and (3) it will provide a hardcopy printout on the printer.

As with an inventory program, the opportunity to make changes exists. This program will allow you to change any note, insert a note after it causing the remaining notes to be moved "down," or delete a note causing the remaining notes to be moved "up."

The program writes the song onto a magnetic file—a RAM file. It can read the song from the RAM file into memory. And the song can be deleted from the RAM file. You will have the opportunity to name your song anything that can fit into six characters.

The program has been written in modules so you may select those modules which seem most appropriate, if you happen not to have enough memory. The program shown will fit within the 32K system. When the song is ended, the melody will linger on—if you don't lose your battery power.

MODULE 1—THE OPENING MENU

This is the opening menu of our program. The options that were mentioned are detailed between statements 80 to 180.

```
10 'OPENING MENU
20 CLEAR 500
30 DIM N(51)
40 DIM T(51)
50 N(1)=9999:T(1)=9999
60 CLS
```

The process begins with a comment line, statement 10. This one happens to be the comment for the OPENING MENU. The comment or REMark line begins with an apostrophe. Statement 20 (CLEAR 500) clears enough memory to get started, though with use you will run out of string space. When you have entered all the program you wish to enter, PRINT FRE(0) and increment this CLEAR statement as far as you can reasonably expect to do so, less a thousand bytes or so.

In this program two arrays are reserved. One is for the frequency assigned to the notes [DIM N(51)] and the other is assigned to the timing [DIM T(51)]. In both cases they have been dimensioned at 51, allowing 50 notes and a sentinel. To begin the table without entries, the first element of each table is loaded with the sentinel in statement 50 [N(1)= 9999:T(1)=9999]. This is an important concept. Every song has a finite number of musical notes which may be numbered in sequence from 1. Yankee Doodle had 32 notes. The design of this program allows 50 or fewer. If this were an inventory situation, you would be restricted to carrying 50 items, which isn't too uncommon for a small business.

In statement 60 we clear the LCD display (CLS) and display the program's menu:

```
70 PRINT "SELECT:";
80 PRINT TAB(10);"<B>UILD SONG";TAB(30);"<S>ING SONG"
90 PRINT TAB(10);"<C>HANGE SONG";TAB(30);"<D>ISPLAY SONG"
100 PRINT TAB(10);"<P>RINT SONG";TAB(30);"<W>RITE SONG"
110 PRINT TAB(10);"<R>EAD SONG";TAB(30);"<K>ILL SONG"
120 PRINT TAB(10);"<E>ND SONG"
```

Operating from a blank screen, nine options are offered on six lines:

1. <B>UILD—The build module permits entry of the song (a maximum of 50 notes) incrementally (note 1, 2, etc.). It helps to have your song planned out on paper before beginning this option.
2. <S>ING—This is the option for playing the song you have entered or have loaded from a file. It couldn't be called *<P>LAY*, as the menu is letter-driven and P will be used for <P>RINT.
3. <C>HANGE—This option allows modification of the song once entered or loaded from media. You may replace a note directly (plus its

timing), you may insert a new note in a specific place, or you may delete a note.

4. <D>ISPLAY—Permits review of the frequencies and timings on the screen.
5. <P>RINT—Permits you to obtain a hardcopy list of the notes and their timings.
6. <W>RITE—Creates a file in RAM where the song may be kept. You may assign the song title.
7. <R>EAD—Loads the song from the RAM file into memory.
8. <K>ILL—Deletes the RAM file from memory. Although not a pretty name, it's descriptive; it's the BASIC command.
9. <E>ND—Returns you to the opening menu.

Here is the coding:

```
130 Z$ = INKEY$:IF Z$ = "" THEN 130
140 IF (Z$="B") OR (Z$="b") THEN GOSUB 230:GOTO 60
150 IF (Z$="S") OR (Z$="s") THEN GOSUB 450:GOTO 60
160 IF (Z$="C") OR (Z$="c") THEN GOSUB 550:GOTO 60
170 IF (Z$="D") OR (Z$="d") THEN GOSUB 1180:GOTO 60
180 IF (Z$="P") OR (Z$="p") THEN GOSUB 1340:GOTO 60
190 IF (Z$="W") OR (Z$="w") THEN GOSUB 1500:GOTO 60
200 IF (Z$="R") OR (Z$="r") THEN GOSUB 1630:GOTO 60
210 IF (Z$="K") OR (Z$="k") THEN GOSUB 1760:GOTO 60
220 IF (Z$="E") OR (Z$="e") THEN MENU
```

Statement 130 (the use of the keyboard strobe, INKEY$) remains operative until a key is pressed. The second instruction in the statement tests for the presence of a null (" "). If a null is present (i.e., nothing has been pressed) the program will remain at statement 130. You can add an instruction at 135 (IF Z$ = CHR$(27) THEN MENU) which will cause return to the menu if the ESCape key has been pressed. That instruction is used other places in the program; it is not necessary here. You might also add an instruction at 225 which will return you to the head of the menu (GOTO 130) if an incorrect key has been pressed.

The IF commands of statements 140 through 220 are known as *compound conditionals,* i.e., two tests are made in the instruction, joined by a conjunction. In this case, the conjunction is OR, which means that only one IF test need be satisfied for the GOSUB action to be taken. As can be seen, the test is for either the upper case or lower case of the anticipated letter response. The module which corresponds to the response is invoked by the GOSUB. The module itself will be terminated by a RETURN, which will return the computer to that portion of the instruction which contains the GOTO 60, at which time the unconditional branch (GOTO) will be taken.

THE BUILD MODULE

The first module discussed is the Build module. Recall that this module will allow direct keyboard entry of your song. You will be entering the notes and the timings directly to their corresponding arrays. You will be asked for the notes sequentially from the first note. Do not be concerned because the notes are in one array and the timings are in another. Their position within their respective arrays is the same.

This is the code:

```
230 ' BUILD MODULE
240 CLS
250 INPUT "HOW MANY NOTES: ";NN
260 IF NN < 51 THEN 280
270 BEEP:PRINT "TOO MANY":GOTO 250
280 CLS
290 FOR X = 1 TO NN
300     PRINT "ENTERING NOTE: ";X
310     INPUT "ENTER FREQUENCY: ";N(X)
320     INPUT "ENTER BEATS IN DECIMAL: ";B
330     T(X) = 40*B
340     PRINT "FREQUENCY:";N(X);"HERTZ"
350     PRINT "BEATS:";B;" LENGTH:";T(X)
360     PRINT "CORRECT (Y/N)?"
370     Z$ = INKEY$:IF Z$ = "" THEN 370
380     IF (Z$="N") OR (Z$="N") THEN 300
390     IF (Z$="Y") OR (Z$="y") THEN 420
400     IF Z$=CHR$(27) THEN 440
410     GOTO 360
420     (X+1)=9999:T(X+1)=9999
430 NEXT X
440 RETURN
```

We begin by clearing the screen (240), asking how many notes are in the song (250), storing the count in the variable NN, and checking to ensure that not more than 50 notes have been requested (260). If more than 50 have been requested, an error condition exists. The operator is BEEPed (270), a message is given, and the request is repeated. If the request is fewer than 51, the process may continue at 280, where the screen is cleared.

Statements 290 and 430 comprise the FOR. . .NEXT loop which will function until the value of NN (Number of Notes) has been achieved. Statement 300 will advise you which note is being entered (X is the value of the current note).

Statements 310 through 330 load the table with the frequency as derived from Table 6-1. The value of B in statements 320 and 330 is the number of beats the note would be, given in decimal. A quarter note would have a response of .25. A dotted half note would have a response of .75. A whole note would have a response of 1.0, etc. Whereas the frequency is loaded directly to the table, the beat is multiplied by 40 be-

fore it is loaded to the table. Recall that a value of 50 equals one second. This value, 40, is set for what is known as *andante*. If you would like to be more sophisticated, establish a variable and change the value with a new question.

Once entered, and even though they have already been loaded to the table, the frequency, beats, and decimal length are displayed for confirmation. In this case, the fact that they have been loaded is of no consequence, as the input process will be repeated if an error was made. In statement 400, the option to abort the entire process via the ESCape key is given. And finally, in statement 420, once the entry has been affirmed, the sentinel is placed in the next position. The sentinel, then, slides along one position ahead of the note being entered.

THE SING MODULE

Calling this module the Sing module is certainly inaccurate, but it will play the song you have entered or, as will be seen later, will play the song you have loaded from a RAM file.

Here is the coding:

```
450 'SING MODULE
460 CLS:PRINT "I'M PLAYING YOUR SONG"
470 FOR X = 1 TO 50
480     IF N(X) = 9999 THEN L = X - 1:GOTO 510
490 NEXT X
500 L = 50
510 FOR X = 1 TO L
520     SOUND N(X),T(X)
530 NEXT X
540 RETURN
```

The instructions between statements 470 and 490 merely determine how long the song is. We are searching for the sentinel. When the sentinel has been located, the value of X has incremented one step beyond the length of the song, so the length variable (L) is given the value of X less 1. If it happens that we go entirely through the array, then L is given a value of 50. The variable L then becomes the controlling limit to the loop which will play the music (statements 510 to 530). Could the two loops have been combined? Yes, we could have looked for the sentinel while playing the song. But if we did that, we wouldn't have been able to show you how to measure the length of an array and use that length for other purposes.

THE CHANGE MODULE

The Change module is by far the most complex module of the program. Essentially it must perform three functions: (1) change an existing note in place; (2) insert a note after a specific place, requiring the rest of the song to be moved down; and (3) delete a note, requiring the rest of the song to be moved up.

The process begins with the following submenu:

```
550 'CHANGE MODULE
560 CLS
570 PRINT "SELECT:"
580 PRINT TAB(10);"<C>CHANGE EXISTING NOTE"
590 PRINT TAB(10);"<I>NSERT A NOTE AFTER"
600 PRINT TAB(10);"<D>ELETE A NOTE"
610 Z$ = INKEY$:IF Z$ = "" THEN 610
620 IF (Z$="C") OR (Z$="c") THEN 670
630 IF (Z$="I") OR (Z$="i") THEN 850
640 IF (Z$="D") OR (Z$="d") THEN 1080
650 GOTO 560
660 CLS
```

There is nothing new or unique in this submenu, so we'll go straight to the first routine, the change to an existing note. Here is the coding:

```
670 INPUT "WHICH NOTE TO CHANGE: ";WN
680 INPUT "ENTER FREQUENCY: ";N(X)
690 INPUT "ENTER BEATS IN DECIMAL: ";B
700 T(X) = 40*B
710 PRINT "FREQUENCY:";N(X);"HERTZ"
720 PRINT "BEATS:";B;" LENGTH:";T(X)
730 PRINT "CORRECT (Y/N)?"
740 Z$ = INKEY$:IF Z$ = "" THEN 740
750 IF (Z$="N") OR (Z$="N") THEN 650
760 IF (Z$="Y") OR (Z$="y") THEN 780
770 IF Z$=CHR$(27) THEN 840
780 PRINT "MORE CHANGES (Y/N)?"
790 Z$ = INKEY$:IF Z$ = "" THEN 790
800 IF (Z$="N") OR (Z$="N") THEN 650
810 IF (Z$="Y") OR (Z$="y") THEN 840
820 IF Z$=CHR$(27) THEN 840
830 GOTO 780
840 RETURN
```

Naturally, this looks just like the input process. The same questions are asked and the array is directly affected. You are given the opportunity to confirm. If you do not confirm you are asked to change it again, and it has already been changed once. This routine does not hold the change for confirmation prior to making the direct change. In another routine that must be done, but not here. An ESCape option is offered.

Once the change is made, you are asked if more changes are to be made. A positive response will keep you in this portion of the module so long as direct changes are to be made. An ESCape option is offered here also.

Note the RETURN instruction at 840. This is the common RETURN for the entire change module. Each of the three routines will use it. It is good practice to have only one RETURN statement, in case changes will be made to the program.

The next routine is the insert routine. The logic of the routine is that you must designate the note *after which* you wish to make entry. This is the coding:

```
850 CLS
860 INPUT "INSERT AFTER WHICH NOTE:";WN
870 IF (WN < 1) OR (WN > 51) THEN BEEP:GOTO 850
880 INPUT "ENTER FREQUENCY: ";F
890 INPUT "ENTER BEATS IN DECIMAL: ";B
900 TT = 40*B
910 PRINT "FREQUENCY:";F;"HERTZ"
920 PRINT "BEATS:";B;" LENGTH:";TT
930 PRINT "CORRECT (Y/N)?"
940 Z$ = INKEY$:IF Z$ = "" THEN 940
950 IF (Z$="N") OR (Z$="N") THEN 850
960 IF (Z$="Y") OR (Z$="y") THEN 990
970 IF Z$=CHR$(27) THEN 840
980 GOTO 940
990 N(51) = 9999:T(51) = 9999
1000 FOR X = 50 TO WN STEP -1
1010    N(X) = N(X-1)
1020    T(X) = T(X-1)
1030 NEXT X
1040 N(WN) = F:T(WN) = TT
1050 PRINT "ADJUSTMENT COMPLETE"
1060 FOR Z = 1 TO 500:NEXT Z
1070 GOTO 840
```

Here the frequency and timing entries are made and confirmed before the new note is inserted. To insert the note, everything from that position on must be shifted "downward" (to a higher value) in the table. Do this by going to the end of the table and moving backwards. This is STEP−1 of statement 1000. The instructions at 1010 and 1020 require explanation. In essence, position 2 receives the value from position 1. Position 3 receives the value from position 2, etc.

New sentinels are placed in 990, in case there is an overrun. From that point on, things are shifted downward one by one, and then the new note and the new timing are inserted. Look at the instructions at statements 1050 and 1060. After the routine is completed and the RETURN is taken, the screen will be cleared. This Z loop merely delays the process long enough for the message to be read.

Finally, there is the routine to delete a note. When a note is deleted, the rest of the song must be shifted upward into the void. The shifting is done from the point of deletion to the end of the table. Here is that coding:

```
1080 CLS
1090 INPUT "WHICH NOTE TO DELETE";WN
1100 IF (WN < 1) OR (WN > 50) THEN BEEP:GOTO 1080
1110 FOR X = WN TO 51
1120     N(X) = N(X+1)
1130     T(X) = T(X+1)
1140 NEXT N
1150 PRINT "ADJUSTMENT COMPLETE"
1160 FOR Z = 1 TO 500:NEXT Z
1170 GOTO 840
```

The instructions at 1120 and 1130 require explanation. In essence, position 1 receives the value from position 2. Position 2 receives the value from position 3, etc.

THE DISPLAY MODULE

The Display module—and the Print module which follows—are really the same module. With it, you can see what has been entered, by passing the table by the screen, one element at a time, with enough detail to review the song. It isn't absolutely necessary, of course, but we begin by determining the length of the table, warning you of the length, advising you to keep a finger on the PAUSE button, and telling you to push any key when you're ready. If that key happens to be the ESCape key, the process is aborted.

Here is the coding:

```
1180 'DISPLAY MODULE
1190 CLS
1200 FOR X = 1 TO 50
1210     IF N(X) = 9999 THEN L = X:GOTO 1240
1220 NEXT X
1230 L = 50
1240 PRINT "YOUR SONG HAS";L;"NOTES"
1250 PRINT "PREPARE TO PAUSE"
1260 PRINT "PRESS ANY KEY TO CONTINUE"
1270 Z$ = INKEY$:IF Z$ = "" THEN 1270
1280 IF Z$ = CHR$(27) THEN 1330
1290 CLS
1300 FOR X = 1 TO L
1310     PRINT "NOTE";X;" ";N(X);" ";"TIME";T(X)
1320 NEXT X
1330 RETURN
```

THE PRINT MODULE

The only difference between this routine and the previous routine is that the printing is done on the hardcopy printer, not on the screen. Otherwise, it is the same routine.

Here is the coding:

```
1340 'PRINT MODULE
1350 CLS
1360 FOR X = 1 TO 50
1370     IF N(X) = 9999 THEN L = X - 1:GOTO 1400
1380 NEXT X
1390 L = 50
1400 PRINT "YOUR SONG HAS";L;"NOTES"
1410 PRINT "PREPARE TO PRINT"
1420 PRINT "PRESS ANY KEY TO CONTINUE"
1430 Z$ = INKEY$:IF Z$ = "" THEN 1430
1440 IF Z$ = CHR$(27) THEN 1490
1450 CLS
1460 FOR X = 1 TO L
1470     LPRINT "NOTE";X;" ";N(X);" ";"TIME";T(X)
1480 NEXT X
1490 RETURN
```

THE WRITE MODULE

When we get to files, some new concepts must be learned. To begin with, there are several types of files. RAM files are files which are created in memory. This program uses RAM files. Then, there are CASsette files. Next, there are MoDeM (MDM) files, which will be used for direct connection communications. And then there are COMmunication files which are used with acoustic coupler communications services and with output destined to a serial printer through the RS-232 connection.

Any file must be assigned to a buffer. A buffer is a portion of memory which is used to remove the data from the program and to compensate for the speed of an external device. For example, the direct connect modem functions at 300 bits per second. The MEW functions much more quickly than that. Thus the need for the buffer. The Model 100 assumes that any input/output will occur along the buffer and assigns a default value of 1, which is the number of buffers in use and the buffer number to be used. To specify the need for another buffer, you must use the MAXFILES statement (e.g., MAXFILES = 2).

Does that mean that if you have two files, you must have a MAXFILES = 2 statement? Not necessarily. The restriction is the number of files which are OPEN at one time. To use files, you must OPEN them before using them, and CLOSE them when you have finished. You can open a file at the beginning of your program and close it at the end, if

you are certain that no other demand will be made upon the file. Or you may open and close them with each transaction. In this program, we have chosen the former.

The form of the OPEN statement is:

OPEN *"device:filename(or configuration)"* FOR *mode* AS *file number*

1. Device will be RAM, CAS, MDM, or COM. It may also be LCD for output to the screen or LPT for output to the printer. In both the latter cases, mode must be OUTPUT.
2. Filename will be a maximum of six characters. The document created in a program will have the suffix .DO, assigned by the system. The suffix need not be specifically designated for most situations. However, it is required for KILL operations. Now, the parenthetical expression "or configuration" is important. It is possible to manufacture by concatenation an expression which will fit in this first operand of the OPEN statement. That is the approach taken in this program, and it will be explained after the coding is shown.
3. Mode can be one of these:
   a. OUTPUT allows data to be written sequentially to the file, starting at the beginning of the file. Despite the fact that Random Access Memory (RAM) is used, there is no way to perform random access writes upon files in the MEW.
   b. INPUT allows data to be read sequentially from the file, starting at the beginning of the file. Again, there is no way to perform random access reads upon files in the MEW.
   c. APPEND simply adds to the length of the file. The file is first read to where the end of file (EOF) marker is found, and then new records are added.
   d. File number will be the number of the buffer which is used. If you have not specified MAXFILES, you must use buffer 1, as it is the default.

Now examine the coding:

```
1500 'WRITE SONG TO RAM FILE
1510 CLS
1520 INPUT "SPECIFY SONG NAME (6 CHAR): ";SN$
1530 IF (LEN(SN$)<1)) OR (LEN(SN$)>6) THEN BEEP:GOTO 1510
1540 FN$ = "RAM:" + SN$
1550 OPEN FN$ FOR OUTPUT AS 1
1560 FOR X = 1 TO 51
1570      PRINT #1,N(X)
1580      PRINT #1,T(X)
1590      IF N(X) = 9999 THEN 1610
1600 NEXT X
1610 CLOSE 1
1620 RETURN
```

The CLOSE of statement 1610 simply specifies the buffer used.

Statement 1520 is the first of the instructions designed to tailor the access to the RAM file. A name is requested (1520); it is tested to ensure that it is not longer than six characters or that the ENTER key hasn't been pressed (by checking for a length less than 1) in statement 1530. This instruction contains the LENgth command. Purpose of the LEN command is to return the length of a string variable. That length can be stored in a numeric variable, e.g., A = LEN(A$), or it can be a part of an IF statement, as shown. Once it is certain that the name of the song (SN$) file falls between one and six characters, the concatenation process may begin.

Statement 1540 FN$ = "RAM:" + SN$ provides the concatenation necessary to identify a RAM file suitable for the OPEN statement. On an output process it is not necessary to specify the extension: The extension .DO is added. In this instance, the string variable FN$ is the configuration mentioned in our discussion of the OPEN statement, which now looks like this: OPEN FN$ FOR OUTPUT AS 1.

As mentioned, file handling on the MEW is sequential. Therefore, the instructions from 1560 to 1600 perform the sequential output. The PRINT # instruction writes a sequential record onto the media. In this example, two successive PRINT # instructions write the note, followed by the timing onto the file. Note that the sentinels are written onto the file before the data are tested for their presence as a means to terminate the write. Thereafter the file is closed and the module is finished.

One of the reasons for placing the sentinel onto the media is to use a file for more than one purpose, separated by a sentinel record. In this instance, under normal circumstances, to write the sentinel onto the media would not be necessary, as an end-of-file indicator is written onto the media in the last position. That end-of-file indicator may be tested with an end-of-file instruction which would be used in the read module (IF EOF(1) THEN . . .).

If this were a CASsette file, nothing would have changed, other than the need to establish the operating controls of the recorder.

Communication files are somewhat more complex, in that the configuration must specify the communications details in the form of the five-character string *rwpbs*, where:

r is the baud rate, a number of 1 to 9 (1 = 75; 2 = 110; 3 = 300; 4 = 600; 5 = 1200; 6 = 2400; 7 = 4800; 8 = 9600; and 9 = 19200). Recall that this feature uses the communications (RS-232) connector.

w is the word length, 6, 7, or 8 bits.

p is the parity, where E = even; O = odd; I = ignore; and N = none.

b is the number of stop bits: 1 or 2.

x is the XON/XOFF status, where E = enable; D = disable.

The mode of a COMmunications file can be either INPUT or OUTPUT.

Modem files (MDM) are communication files which are transmitted using the direct connect modem. Their configuration consists of the four-character string, wpbs. The baud rate, you will recall, is fixed at 300.

Screen files (LCD) are OUTPUT mode only. No configuration is necessary. Why is a screen file necessary? Whereas the PRINT statement writes a character line at a time, and the graphics functions show characters and lines, the screen file can fill an entire screen nearly instantaneously, a fine way to accomplish rapid screen change.

Printer files (LPT) are OUTPUT mode only. No configuration is necessary. Why is a printer file necessary? The printer isn't going to operate any faster than it can, and will place no more characters on a line than it can physically do so. Perhaps the greatest value is to format a file for the most rapid transfer to the print buffer.

THE READ MODULE

The purpose of this module is simply to read the file from the RAM file and to install it into the array. The OPEN statement has been changed to indicate INPUT. Where the file was written by using PRINT #, it is now read by using INPUT #. A new part of the file name has been concatenated—the extension .DO, for document, required to distinguish the file from others. Other than these things, this routine looks similar to the others.

```
1630 'READ SONG FROM RAM FILE
1640 CLS
1650 INPUT "SPECIFY SONG NAME (6 CHAR): ";SN$
1660 IF (LEN(SN$)<1)) OR (LEN(SN$)>6) THEN BEEP:GOTO 1640
1670 FN$ = "RAM:" + SN$ + ".DO"
1680 OPEN FN$ FOR INPUT AS 1
1690 FOR X = 1 TO 51
1700      INPUT #1,N(X)
1710      INPUT #1,T(X)
1720      IF N(X) = 9999 THEN 1740
1730 NEXT X
1740 CLOSE 1
1750 RETURN
```

Recall that the EOF command could have been used instead of the test for sentinel. Note that the sentinel is tested *after* the transfer to the array. The EOF instruction was used in this instance simply to avoid having to separately store the sentinel, no other reason.

THE KILL (FILE DELETION) MODULE

This module is the easiest one in the program. The file name is requested, as before, and is compounded to the device (RAM) and the extension (.DO) to create a complete filename (FN$), which is then deleted by the KILL instruction:

```
1760 'KILL SONG ON RAM FILE
1770 CLS
1780 INPUT "SPECIFY SONG NAME (6 CHAR): ";SN$
1790 IF (LEN(SN$)<1)) OR (LEN(SN$)>6) THEN BEEP:GOTO 1770
1800 FN$ = "RAM:" + SN$ + ".DO"
1810 KILL FN$
1820 RETURN
```

Here is the program in its entirety:

```
10 'OPENING MENU
20 CLEAR 500
30 DIM N(51)
40 DIM T(51)
50 N(1)=9999:T(1)=9999
60 CLS
70 PRINT "SELECT:";
80 PRINT TAB(10);"<B>UILD SONG";TAB(30);"<S>ING SONG"
90 PRINT TAB(10);"<C>HANGE SONG";TAB(30);"<D>ISPLAY SONG"
100 PRINT TAB(10);"<P>RINT SONG";TAB(30);"<W>RITE SONG"
110 PRINT TAB(10);"<R>EAD SONG";TAB(30);"<K>ILL SONG"
120 PRINT TAB(10);"<E>ND SONG"
130 Z$ = INKEY$:IF Z$ = "" THEN 130
140 IF (Z$="B") OR (Z$="b") THEN GOSUB 230:GOTO 60
150 IF (Z$="S") OR (Z$="s") THEN GOSUB 450:GOTO 60
160 IF (Z$="C") OR (Z$="c") THEN GOSUB 550:GOTO 60
170 IF (Z$="D") OR (Z$="d") THEN GOSUB 1180:GOTO 60
180 IF (Z$="P") OR (Z$="p") THEN GOSUB 1340:GOTO 60
190 IF (Z$="W") OR (Z$="w") THEN GOSUB 1500:GOTO 60
200 IF (Z$="R") OR (Z$="r") THEN GOSUB 1630:GOTO 60
210 IF (Z$="K") OR (Z$="k") THEN GOSUB 1760:GOTO 60
220 IF (Z$="E") OR (Z$="e") THEN MENU
230 ' BUILD MODULE
240 CLS
250 INPUT "HOW MANY NOTES: ";NN
260 IF NN < 51 THEN 280
270 BEEP:PRINT "TOO MANY":GOTO 250
280 CLS
290 FOR X = 1 TO NN
300     PRINT "ENTERING NOTE: ";X
310     INPUT "ENTER FREQUENCY: ";N(X)
320     INPUT "ENTER BEATS IN DECIMAL: ";B
330     T(X) = 40*B
340     PRINT "FREQUENCY:";N(X);"HERTZ"
350     PRINT "BEATS:";B;" LENGTH:";T(X)
360     PRINT "CORRECT (Y/N)?"
370     Z$ = INKEY$:IF Z$ = "" THEN 370
380     IF (Z$="N") OR (Z$="N") THEN 300
390     IF (Z$="Y") OR (Z$="y") THEN 420
```

```
400     IF Z$=CHR$(27) THEN 440
410     GOTO 360
420     (X+1)=9999:T(X+1)=9999
430 NEXT X
440 RETURN
450 'SING MODULE
460 CLS:PRINT "I'M PLAYING YOUR SONG"
470 FOR X = 1 TO 50
480     IF N(X) = 9999 THEN L = X - 1:GOTO 510
490 NEXT X
500 L = 50
510 FOR X = 1 TO L
520     SOUND N(X),T(X)
530 NEXT X
540 RETURN
550 'CHANGE MODULE
560 CLS
570 PRINT "SELECT:"
580 PRINT TAB(10);"<C>CHANGE EXISTING NOTE"
590 PRINT TAB(10);"<I>NSERT A NOTE AFTER"
600 PRINT TAB(10);"<D>ELETE A NOTE"
610 Z$ = INKEY$:IF Z$ = "" THEN 610
620 IF (Z$="C") OR (Z$="c") THEN 670
630 IF (Z$="I") OR (Z$="i") THEN 850
640 IF (Z$="D") OR (Z$="d") THEN 1080
650 GOTO 560
660 CLS
670 INPUT "WHICH NOTE TO CHANGE: ";WN
680 INPUT "ENTER FREQUENCY: ";N(X)
690 INPUT "ENTER BEATS IN DECIMAL: ";B
700 T(X) = 40*B
710 PRINT "FREQUENCY:";N(X);"HERTZ"
720 PRINT "BEATS:";B;" LENGTH:";T(X)
730 PRINT "CORRECT (Y/N)?"
740 Z$ = INKEY$:IF Z$ = "" THEN 740
750 IF (Z$="N") OR (Z$="N") THEN 650
760 IF (Z$="Y") OR (Z$="y") THEN 780
770 IF Z$=CHR$(27) THEN 840
780 PRINT "MORE CHANGES (Y/N)?"
790 Z$ = INKEY$:IF Z$ = "" THEN 790
800 IF (Z$="N") OR (Z$="N") THEN 650
810 IF (Z$="Y") OR (Z$="y") THEN 840
820 IF Z$=CHR$(27) THEN 840
830 GOTO 780
840 RETURN
850 CLS
860 INPUT "INSERT AFTER WHICH NOTE:";WN
870 IF (WN < 1) OR (WN > 51) THEN BEEP:GOTO 850
880 INPUT "ENTER FREQUENCY: ";F
890 INPUT "ENTER BEATS IN DECIMAL: ";B
900 TT = 40*B
910 PRINT "FREQUENCY:";F;"HERTZ"
920 PRINT "BEATS:";B;" LENGTH:";TT
930 PRINT "CORRECT (Y/N)?"
940 Z$ = INKEY$:IF Z$ = "" THEN 940
950 IF (Z$="N") OR (Z$="N") THEN 850
960 IF (Z$="Y") OR (Z$="y") THEN 990
970 IF Z$=CHR$(27) THEN 840
980 GOTO 940
990 N(51) = 9999:T(51) = 9999
```

```
1000 FOR X = 50 TO WN STEP -1
1010    N(X) = N(X-1)
1020    T(X) = T(X-1)
1030 NEXT X
1040 N(WN) = F:T(WN) = TT
1050 PRINT "ADJUSTMENT COMPLETE"
1060 FOR Z = 1 TO 500:NEXT Z
1070 GOTO 840
1080 CLS
1090 INPUT "WHICH NOTE TO DELETE";WN
1100 IF (WN < 1) OR (WN > 50) THEN BEEP:GOTO 1080
1110 FOR X = WN TO 51
1120     N(X) = N(X+1)
1130     T(X) = T(X+1)
1140 NEXT N
1150 PRINT "ADJUSTMENT COMPLETE"
1160 FOR Z = 1 TO 500:NEXT Z
1170 GOTO 840
1180 'DISPLAY MODULE
1190 CLS
1200 FOR X = 1 TO 50
1210     IF N(X) = 9999 THEN L = X:GOTO 1240
1220 NEXT X
1230 L = 50
1240 PRINT "YOUR SONG HAS";L;"NOTES"
1250 PRINT "PREPARE TO PAUSE"
1260 PRINT "PRESS ANY KEY TO CONTINUE"
1270 Z$ = INKEY$:IF Z$ = "" THEN 1270
1280 IF Z$ = CHR$(27) THEN 1330
1290 CLS
1300 FOR X = 1 TO L
1310     PRINT "NOTE";X;" ";N(X);" ";"TIME";T(X)
1320 NEXT X
1330 RETURN
1340 'PRINT MODULE
1350 CLS
1360 FOR X = 1 TO 50
1370     IF N(X) = 9999 THEN L = X - 1:GOTO 1400
1380 NEXT X
1390 L = 50
1400 PRINT "YOUR SONG HAS";L;"NOTES"
1410 PRINT "PREPARE TO PRINT"
1420 PRINT "PRESS ANY KEY TO CONTINUE"
1430 Z$ = INKEY$:IF Z$ = "" THEN 1430
1440 IF Z$ = CHR$(27) THEN 1490
1450 CLS
1460 FOR X = 1 TO L
1470     LPRINT "NOTE";X;" ";N(X);" ";"TIME";T(X)
1480 NEXT X
1490 RETURN
1500 'WRITE SONG TO RAM FILE
1510 CLS
1520 INPUT "SPECIFY SONG NAME (6 CHAR): ";SN$
1530 IF (LEN(SN$)<1)) OR (LEN(SN$)>6) THEN BEEP:GOTO 1510
1540 FN$ = "RAM:" + SN$
1550 OPEN FN$ FOR OUTPUT AS 1
1560 FOR X = 1 TO 51
1570     PRINT #1,N(X)
1580     PRINT #1,T(X)
1590     IF N(X) = 9999 THEN 1610
```

```
1600 NEXT X
1610 CLOSE 1
1620 RETURN
1630 'READ SONG FROM RAM FILE
1640 CLS
1650 INPUT "SPECIFY SONG NAME (6 CHAR): ";SN$
1660 IF (LEN(SN$)<1)) OR (LEN(SN$)>6) THEN BEEP:GOTO 1640
1670 FN$ = "RAM:" + SN$ + ".DO"
1680 OPEN FN$ FOR INPUT AS 1
1690 FOR X = 1 TO 51
1700      INPUT #1,N(X)
1710      INPUT #1,T(X)
1720      IF N(X) = 9999 THEN 1740
1730 NEXT X
1740 CLOSE 1
1750 RETURN
1760 'KILL SONG ON RAM FILE
1770 CLS
1780 INPUT "SPECIFY SONG NAME (6 CHAR): ";SN$
1790 IF (LEN(SN$)<1)) OR (LEN(SN$)>6) THEN BEEP:GOTO 1770
1800 FN$ = "RAM:" + SN$ + ".DO"
1810 KILL FN$
1820 RETURN
```

You now have a solid idea of how it is possible to develop files in arrays and to write them to a magnetic media (such as a RAM file) and to read them back in. It's time now to add to your level of sophistication, for the handling of files has some rationale—files exist for a reason and for a set of processing considerations. The first of these is time itself.

No Inclination—No TIME$

Earlier in the book, we presented a small BASIC program which was an alarm clock. Recall that the program BEEPed enough times to awaken a light sleeper. We didn't do it then, but several other instructions would have been useful. They are:

TIME$ ON/OFF/STOP

and

ON TIME$ GOSUB

These begin our discussion of *interrupts*. Recall that in the alarm clock program the instruction was written in the form: IF TIME$ = "07:00:00" GOTO . . . a specific statement number. That assumed that the program kept checking the clock to see what the time was. The program was written that way.

Suppose for the sake of argument that the program was designed for some other purpose and you didn't want to build a test for the time into every loop within the program. What would you do? You would build in an "ON" test—in this case, the ON TIME$ GOSUB. The routine would be included at the beginning of the program and would be a statement of condition of the time. The MEW would then constantly check the time for you and, when the appropriate time was noted, would branch to the specified subroutine. That subroutine would be terminated by a RESUME statement, rather than the RETURN which has been used up to this point.

All is not quite so simple, however. The ON test works only if the clock has been *enabled.* Enabled merely means "activated—turned on." The machine comes on in the enabled state. The three forms of the TIME$ instruction are:

1. TIME$ ON enables ON TIME$ interruptions.
2. TIME$ OFF disables ON TIME$ interruptions.
3. TIME$ STOP disables the interrupt. If you have not specified otherwise, the MEW performs a time interrupt, whether or not you use it in an ON TIME$ expression. This command allows the MEW to bypass the TIME$ test altogether.

This is the alarm clock problem shown previously:

```
10 CLS
20 PRINT TIME$
30 IF TIME$ = "07:00:00" THEN 50
40 GOTO 10
50 FOR N = 1 TO 500
60     PRINT CHR$(7)
70 NEXT N
80 END
```

Using the TIME$ commands just discussed, the program would be written this way:

```
10 ON TIME$ = "07:00:00" GOSUB 50:END
20 CLS
30 PRINT TIME$
40 GOTO 20
50 FOR N = 1 TO 500
60     BEEP
70 NEXT N
80 RESUME
```

And BEEP is the same as PRINT CHR$(7).

Errors

The same concept is used when treating errors. Part of the problem with reference manuals are the terms which are given, but left undefined. Nowhere is this more evident than with error codes. While some codes are self-explanatory, others leave you wondering what happened and what they mean. These are the errors which you could encounter when using the MEW, with a brief explanation of why they might occur:

CODE	MESSAGE	MEANING
1	NF	NEXT without FOR. You have probably jumped into the middle of a FOR . . . NEXT loop.
2	SN	Syntax Error. You have violated the rules of an instruction at that statement address.
3	RG	RETURN without GOSUB. You have entered a subroutine by other than the normal means.
4	OD	Out of Data. You were READing DATA lines and you executed one READ too many for the number of DATA elements contained in the DATA lines.
5	FC	Illegal Function Call. You tried to do something which is physically impossible, such as RESTORE the modem.
6	OV	Overflow. You tried to develop too large a number for the space which could hold it.
7	OM	Out of Memory. You have exceeded the available memory. You must reduce the size of your program.
8	UL	Undefined Line. You branched to a statement that doesn't exist. This usually happens when you remove an instruction without ensuring that all references to it are adjusted.
9	BS	Bad Subscript. You have attempted to reference an array element which lies beyond that which was DIMensioned.
10	DD	Doubly Dimensioned Array. You tried to DIMension the same array twice.
11	/0	Division by Zero. This is most often caused when dividing by a value given to the computer as a result of an INPUT statement and the operator simply pressed the ENTER key.
12	ID	Illegal Direct. In Direct mode you attempted a command which is only meaningful when done within a program, such as DIMensioning.

13	TM	Type Mismatch. You tried to do something with mixed variable types. Perhaps you tried to set a numeric variable equal to a string variable.
14	OS	Out of String Space. This is caused by exceeding the default string space size (256 bytes) or the size specified in the CLEAR statement. Remember that often string variables are not defined until the program is executed. Go back and increment your CLEAR statement (or use one, if you had not done so), keeping in mind the available memory of your program.
15	LS	String Too Long. You may never see this one if you work within the size of your screen. It simply means you exceeded the maximum length permitted for strings—255 bytes.
16	ST	String Formula Too Complex. In simple terms, you have confused the MEW. Break the instruction down into smaller, more easily understood instructions.
17	CN	Can't Continue. CONTinue only works where a STOP has been executed. If your program stopped for any other reason, and you try to CONTinue, you will receive this error.
18	IO	Input/Output Error. Something isn't right with an I/O operation. Perhaps you have used up all the permitted file names. Perhaps you tried to print a file to a hardcopy printer and that printer was not turned on or ready.
19	NR	No RESUME. You have entered a subroutine via the interruption mechanism (ON . . .). You've gotten to the end of the routine and there is no way to get back. This is most often the result of ending interruption routines with RETURN.
20	RW	RESUME without Error. You got into an interrupt handler without coming there by the ERROR routine which would have triggered the interrupt. This is often the result of treating the interrupt handler like a normal subroutine.
21	UE	Undefined Error. This error code is not used.
22	MO	Missing Operand. Some part of the anticipated instruction is missing. This frequently happens when the length attributes are omitted from the LEFT$, RIGHT$, and MID$ functions.
23–49	UE	Undefined Error. These error codes are not used.
51	BN	Bad File Number. You used buffer 2 or above without telling the computer via MAXFILES.
52	FF	File Not Found. You requested a file that was not available.
53	AO	Already Open. You tried to open a file which was already opened. Typical reason is restarting a program before a CLOSE has been performed. If the program is restarted, CLOSE the file via direct command.

54	EF	Input Past End of File. No End-of-File (EOF) marker was on the media. This usually happens when an attempt is made to read a file whose writing had been interrupted. The only way to avoid this is to modify the program to count the records it reads, note the count when the error occurs, and then read only that number of records.
55	NM	Bad File Name. You've done something to the name of the file which contravenes the rules.
56	DS	Direct Statement in File. LLIST is a direct statement which would not be placed in a program or in a file. Any direct statement will cause havoc if contained in a file, as execution is suspended.
57	FL	Undefined Error. This is a file error which cannot be diagnosed.
58	CF	File Not Open. You tried to read from or write to a file which has not been OPENed.
59–255	UE	Undefined Error. These error codes are not used.

You will not be faced with intercepting and handling all of these errors dynamically, i.e., while the program is running. In fact, you must handle those which have a high probability of occurrence. Because the error can be checked as a function of interrupt, establish the interrupt early in the program and then, in the interrupt routine, establish what to do if the situation occurs. In the following example, assume that variables A, B, and C—defined in the program—would have been received as a function of an INPUT statement.

```
10 CLS
20 ON ERROR GOTO 1000
30 A = 2
40 B = 4
50 C = 0
60 D = (A + B)/C
70 PRINT D
80 END
1000 PRINT "Error: ";ERR;"in line";ERL
1010 IF ERR = 11 THEN C = 1:GOTO 1030
1020 PRINT "Error Unresolved":STOP
1030 RESUME
```

Here we have an example of an error routine which was invoked on the basis of a divide by zero, which was, of course, staged. This common error is caused by accidentally hitting the ENTER key when a value is expected. The ON ERROR set up the interrupt routine to be used (at 1000). In the interrupt routine, you'll note two instructions used

to handle errors: ERR, which is the way to use the actual error code; and ERL which is used to treat the error line. If you knew that the only error you would receive would be at line 60, you could have handled it this way: IF ERL = 60 THEN C = 1. However, that presupposes that you will never remove or revise statement 60.

It is possible to simulate an error merely by including an ERROR statement in the program. Here is the same program revised to force the error. Note that variable C has been changed so the error would not otherwise occur:

```
10 CLS
20 ON ERROR GOTO 1000
30 A = 2
40 B = 4
50 C = 1
55 ERROR 11
60 D = (A + B)/C
70 PRINT D
80 END
1000 PRINT "Error: ";ERR;"in line";ERL
1010 IF ERR = 11 THEN C = 1:GOTO 1030
1020 PRINT "Error Unresolved":STOP
1030 RESUME
```

Communications Interruptions

Two kinds of communications interrupts can occur: one through the RS-232 port and the other through the direct connect modem. They are similar in nature, and presuppose that one or the other is established in anticipation of the receipt of information through the channel. Thereafter, whatever program will be running in the MEW must handle a communication interrupt of the sort anticipated. Again, to use a communications interrupt, it must first be enabled. Enabling is not automatic with communications services. That process is described next:

COM ON/OFF/STOP OR MDM ON/OFF/STOP

1. COM/MDM ON enables ON COM or ON MDM interruptions.
2. COM/MDM OFF disables ON COM or ON MDM interruptions.
3. COM/MDM STOP disables the interrupt. However, even though the process has been disabled, BASIC will remember that something came in over the communications line. Thus, after enabling, if you issue an ON COM command, the communications interrupt handler will be invoked, telling you that you must pursue the communication.

The ON statements will go at the beginning of the program, like this:

```
20 ON COM GOSUB 1000
      or
20 ON MDM GOSUB 1000
```

This assumes that, prior to the statement, either has been enabled. Now, note that the ON COM and ON MDM statements both are followed by GOSUB. That's because once the communications interrupt has been handled, you will return to whatever was being done. That is not necessarily true of the ON ERROR, thus it was followed by GOTO.

The KEYs to the Program

In several programs previously discussed, we tested for the presence of an ESCape key [CHR$(27)] as a means to break out of a routine. It must be used where it was tested and could not be applied in an interrupt manner. However, it is possible to disable the normal function key and assign the interrupt function to one of the eight function keys so that an interrupt routine can be defined and constructed. Naturally, the process begins by enabling:

KEY ON/OFF/STOP

1. KEY ON enables ON KEY interruptions.
2. KEY OFF disables ON KEY interruptions.
3. KEY STOP disables the interrupt. However, even though the process has been disabled, BASIC will remember that something came in over the communications line. Thus, after enabling, if you issue an ON KEY command, the function key interrupt handler will be invoked, telling you that you must pursue the key setting.

You may turn all the keys or a single key on (or off), like this:

```
100 KEY ON
110 KEY (5) OFF
```

At any time you can obtain a list of the keys in use, simply by typing the direct command KEY LIST. You may wish to include that in a program, but there isn't a real reason to do so unless you need periodic reminder of the value of the keys. The list of keys will be displayed in

four rows of two (i.e., 1-2, 3-4, 5-6, and 7-8), and if you have not altered the assigned key functions, the display should be:

```
Files       Load"
Save        Run
List
            Menu
```

The KEY functions can be reset. For instance, once you have started your clock, you can determine the time of day merely by pressing the appropriate function key. The seventh function key isn't assigned, so let's do that:

```
KEY 7, "?TIME$" + CHR$(13)
```

On the surface, this will look erroneous. First, bear in mind that the ? is a substitute for the BASIC command PRINT. Next, and you'll find this only by doing it, the first four characters of whatever is assigned to the function key show in the appropriate place on the screen when the LABEL command key has been pressed.

Next, a problem appears with combining the command and the reserved string variable name in quotations. This simply passes that configuration to the matrix which is used by the function key, and passes it intact. When pressed, it will be correctly interpreted by the BASIC interpretive compiler. The CHR$(13) which has been concatenated is the ASCII code for ENTER. Each function key has a small matrix assigned to it. If you try to compound reserved words (such as DAY$ and DATE$) to TIME$, you will receive a BS error.

Once you have reassigned any function key, you must invoke a machine language program to reestablish the cold start defaults. Therefore, if you do reassign the function keys and wish to restore them, include these two instructions immediately prior to ending the program:

```
CALL 23164,0,23366
CALL 27795
```

These are two assembler language subroutines within the BASIC ROM section which will restore the original function keys. While this book does not teach the assembler language, it is possible to obtain assembler language books for the Model 100's processor chip (the 8085) and to learn the language. Whenever you use an assembler routine, the CALL instruction is necessary.

Before discussing function key interruption handling, we should discuss two noninterrupt instructions: ON. . .GOTO and ON. . .GOSUB. Suppose a variable is to receive a value selected from the keyboard,

such as a menu selection with three possible alternatives. You could test (if it's 1; if it's 2, etc.) or you could take a specific action on the basis of the input:

```
10 CLS
20 PRINT "1.  Option One"
30 PRINT "2.  Option Two"
40 PRINT "3.  Option Three"
50 INPUT A
60 IF (A < 1) OR (A > 3) THEN 10
70 ON A GOTO 100,200,300
80 GOTO 10
```

If the value of A is 1, the program would go to 100; if 2, to 200; if 3, to 300. All other values are bypassed in statement 60. From those addresses, the programmer controls direction. If, instead on the ON. . . GOTO, you had substituted ON. . .GOSUB, it would be necessary to establish a RETURN command at the end of each subroutine.

The ON KEY GOSUB works precisely the same way. It would probably be the first instruction after the KEYs have been enabled:

```
10 KEY OFF

20 ON KEY GOSUB 1000,2000,3000,,,,,8000
```

Each subroutine would be terminated by RESUME. The string of commas are called positional commas, required to place the last in its appropriate position. If Function Keys 4 to 7 were pressed, no error handling would be done.

A few commands, which are important, do not fit neatly into our previous discussions. Yet, they deal with program and data handling.

Scroll

The SCREEN instruction may be used to ensure that the eighth line is available for data or text if scrolling occurs. The two forms of the instruction are:

SCREEN 0,0 causes the LABEL line to disappear.

SCREEN 0,1 causes the LABEL line to reappear.

The latter is the default option. Do not be concerned about the format of this instruction. Microsoft, which devised the language, uses this instruction differently in other microcomputers.

Program Development

Suppose you had developed a complicated set of DATA statements to incorporate into several programs. To ensure that they would never conflict with any other instructions, they should be numbered with a high number, say 5000. You would first load the BASIC interpreter by pressing ENTER when the cursor indicates it is there or by loading any existing BASIC programs. Then, in direct mode, you would type MERGE "RAM:DAT". Don't call it DATA, as that will cause a conflict. But, note that a RAM file has been designated. CASsette files, COMmunication files, or MoDeM files may also be used. The rules for each are similar to other uses we have discussed and are detailed in the reference manual. From that point, the RAM file DAT.DO would be a part of the current program.

To merge programs dynamically, on the basis of a menu option, is also possible. Suppose the programs are too large to fit into the available memory of the MEW. You can then break down the program into separate functions, number the line numbers identically so they would occupy the same space, and merge the program pieces that you need. For example:

```
10 CLS
20 PRINT "1.  Option One"
30 PRINT "2.  Option Two"
40 PRINT "3.  Option Three"
50 INPUT A
60 IF (A < 1) OR (A > 3) THEN 10
70 ON A GOTO 100,200,300
80 GOTO 10
100 MERGE "RAM:P1":GOTO 400
200 MERGE "RAM:P2":GOTO 400
300 MERGE "RAM:P3":GOTO 400
```

RAM

Occasionally changing the name of a file is necessary. The most common reason is to generate another file of the same name. In Direct mode, that can be done like this:

```
NAME "RAM:ABC.DO" AS "RAM:DEF.DO"
```

Since renaming can only occur for RAM files, the specification of "RAM" need not be used.

The process can also be done indirectly, like this:

```
10 INPUT "OLD FILE NAME: ";OF$
20 INPUT "NEW FILE NAME: ";NF$
30 NAME OF$ AS NF$
```

The POWER Function

At some time in your life you're certain to have encountered the electric bank which, when you deposit your coin, begins to operate: a trapdoor opens, a hand emerges, and it shuts off the switch. Believe it. The MEW works like that, with three options oriented to the POWER function. All of them have something to do with shutting down the computer.

The first is the automatic power down option, POWER. Turn on the machine for awhile and you'll see that, if not used, it shuts itself off to conserve the batteries. You can change that, by following POWER with a numeric expression in the range of 10 to 255. That expression is multiplied by one tenth of one minute. POWER 100 is the default (10 minutes).

The second disables the automatic shutdown; you would want to use it only when the MEW is connected to an electric outlet. The command is: POWER CONT.

The third, POWER OFF, shuts down the computer immediately. An added command, RESUME (i.e., POWER OFF,RESUME) will shut off the power but mark its place in the program so that you may continue at that point when the power is reactivated.

The latter two commands are used to inhibit the system from shutting down in anticipation of a communications interrupt and, after the communications have been received, to shut the machine off immediately. Whether you resume processing on power up will depend on how you handle the data which has arrived over the communications media.

Data Conversion

Recall a prior program wherein we set a numeric variable equal to the value of a string variable, e.g., A = VAL(A$), and yet another where the numeric variable was given a string representation, e.g., A$ = STR$(A). These were two forms of data conversion. Other forms of data conversion and modification exist:

CSNG returns a single-precision value.

```
10 A! = CSNG (0.66666666666)
        returns 0.666667 to A!
```

CDBL returns a double precision value.

```
10 A% = 45
20 A# = CDBL(A%)
```

returns 45.0000000000000 TO A#

CINT returns the whole number portion of a number.

```
10 A = 23.45
20 B = CINT(23.45)
30 C = INT(A)
```

When this sequence is done, B and C will have the same value: 23.

Negative Numbers

We discussed the INTeger instruction prior to the CINT discussion. In neither case did we show what happened to negative numbers. When negative numbers are used, the INTeger instruction returns the whole number not greater than the numeric expression. Thus, if the number was a negative value, such as −111.11, the INTeger command would return the value of −112. The FIX instruction, on the other hand, truncates any decimal value. Given −111.11, FIX would return a value of −111.

Machine Address

Now and again the specific machine address for a variable must be located. That is done with the VARPTR instruction, like this: A = VARPTR(A$). The machine address of the string variable is placed in the numeric variable. If the number returned is negative, it merely means a memory address larger than 32767 has been addressed. Such memory addresses are possible within the machine itself.

This address is useful in three specific commands:

CALL—for linking to a machine language routine.

PEEK—for reading a specific address:

```
PRINT PEEK(10110)
```

POKE—for writing a character to a specific address:

```
POKE 10110,10
```

Other Kinds of Input

One of the problems of the INPUT statement is that, while it can accept either numeric or string data, it cannot accept imbedded punctuation.

You will recall that INPUT A,B is really two input instructions which accept two values in succession. The instruction INPUT F$,M$,L$ would allow you to enter your name (First, Middle Initial, Last) in three separate pieces divided by a comma, for example, KENNISTON,W.,LORD. In this example, F$ would have "KENNISTON", M$ would have "W.", and L$ would have "LORD." If you happened to forget the comma, F$ would have KENNISTON W. LORD, and M$ and L$ would have nothing (null). So, you plan to put it all in one variable and advise people not to use commas and along comes someone like the author, whose name contains an imbedded comma ("KENNISTON W. LORD, JR."), and things get all fouled up. The way around that is to accept it to a single variable with the LINE INPUT instruction, like this:

```
10 LINE INPUT "(ENTER YOUR WHOLE NAME)";NM$
```

Now the imbedded comma will make no difference.

Next, recall when we measured the length of a response to ensure that not fewer than one nor more than six characters had been entered. Recall also the INKEY$ instruction which strobed the keyboard for a single character. The concepts are blended with the INPUT$ instruction, available in two forms:

1. INPUT$ (number of characters to be entered):
   FN$ = INPUT$(6) would gain a six-character filename from the keyboard.
2. INPUT$ (number of characters, file number):
   PW$ = INPUT$(8,1) would gain an eight-character password from the head of a file which had been opened as file number one.

And finally, the INP instruction reads a specific byte from an input port in the range of 0 to 255, e.g., A% = INP(6). Gaining and sending information via a port are beyond the scope of this book.

Other Kinds of Output

The output correspondent to the INP command is OUT. The instruction OUT 255,1 would send the byte 1 to output port 255. Again, this is beyond the scope of this book.

However, the PRINT USING sequences are very much within the scope of this book. We discussed PRINT USING before when it was necessary to punctuate some money fields. The command is customarily used to add punctuation to numeric fields, not only to money fields. The

power of the instruction is great, and while we have no desire to duplicate the manual here, it would be useful to summarize the rules and features:

#	This is the fill character. It anticipates the number of positions in the editing mask in which numbers will be placed.
+	This is one of two sign characters. It may be placed at the beginning of the editing mask (i.e., +#####) or at the end of the editing picture (i.e., #####+). If the data which are being placed through the editing mask are positive, no sign will show, though the position will be used. If the data are negative, the minus sign will be shown in the position of the mask occupied by the plus sign.
−	This is the other sign character; the same rules apply here.
**	When placed at the head of the editing mask (i.e., **#####), the asterisks will be filled to the left of all money fields, to the extent of the mask size.
$$	When placed at the head of the editing mask (i.e., $$#####), the dollar sign will be floated from the left side against the money figure.
**$	When placed at the head of the editing mask (i.e., **$$#####), the dollar sign will be floated and the field will be asterisk filled from the left side against the money figure.
(.)	Punctuation may also be part of the edit mask. The editing mask ##,###.##, when used with the data 1234.56 would return 1,234.56.

The same editing mask may be used for multiple fields:

```
PRINT USING "$$##,###.## ";A,B,C
```

And finally, the form of the statement used for writing formatted files is: PRINT # USING. The only difference is the addition of the file number:

```
PRINT #1 USING "$$##,###.## ";A,B,C
```

This would be useful for preparing output report files which will be printed at a later date or for writing to the screen as a file.

Whatever Makes Your MOTOR Run

A specific instruction will turn the cassette motor on and off. It's called, appropriately, MOTOR. Don't stay up all night trying to think up uses for it. The only one I can think of is the ability to key an audio tape to a visual presentation for educational purposes.

Summary

As this chapter closes, let's add the commands, instructions, and functions we've used in it to our list at the end of Chapter 5.

COMMANDS

CLEAR	Clears program variables and dimensioned tables. It is required whenever string variables and arrays occupy more than 256 bytes of memory.
FRE	Determines the available memory.
HIMEM	Gets the High Memory Address.
KEY	Defines the function keys.
KEY LIST	Lists the function key definitions.
KILL	Deletes a RAM file.
LCOPY	Copies the screen to a printer.
MERGE	Combines two BASIC programs.
NAME. . .AS	Renames a file.
RUN	Executes the current BASIC program.
RUNM	Loads and executes a machine language program.
SAVEM	Stores a machine language program.
SCREEN	Locks/unlocks the LABEL line.

NON-I/O STATEMENTS

CDBL	Converts a numeric expression to double precision.
CINT	Converts a numeric expression to an integer.
CSNG	Converts a numeric expression to single precision.
COM ON/OFF/ STOP	Enables/disables the communications interrupt.

DEFtype	Defines default variable types where "type" may be INTeger, SiNGle precision, DouBLe precision, or STRing (INT, SNG, DBL, or STR).
DIM	Allows specification of an array in one or two dimensions. It is required for tables in excess of 10 elements. A zero-relative element exists but must be specifically addressed. This statement declares the maximum subscript values and reserves the space.
FIX	Truncates a value without rounding.
INT	Gets the whole number representation of a variable.
KEY ON/OFF/ STOP	Enables/disables the function key interrupts.
MAXFILES	Specifies the maximum number of files in use.
MAXRAM	Returns the amount of installed memory.
MDM ON/OFF/ STOP	Enables/disables the modem interrupts.
MENU	Returns control to the main menu.
MERGE	Combines two BASIC programs.
ON COM GOSUB	Defines a communications interrupt.
ON ERROR GOTO	Defines an error interrupt.
ON KEY GOSUB	Defines function key interrupts.
ON MDM GOSUB	Defines modem interrupt.
ON TIME$ GOSUB	Defines clock interrupt.
POWER	Automatic Power Down.
POWER CONT	Prevents Automatic Power Down.
POWER OFF	Immediate Power Down.
RESUME	Resumes execution after error or interrupt.
RND	Returns pseudo random number (0-1).
SOUND ON/ OFF	Enables or disables sound.
TIME$ ON/ OFF/STOP	Enables/disables Time Interrupt.
VARPTR	Gets the address of a variable.

I/O STATEMENTS

BEEP	Beeps the speaker—a function also performed by CHR$(7).
CALL	Calls a machine-level subroutine.
CLOAD	Loads a program from cassette tape.
CLOAD?	Loads a program from cassette tape, verifying it against the program stored in memory.
CLOADM	Loads a machine language program from cassette tape.
CLOSE	Closes the file(s) specified.
CSAVE	Saves a program on cassette tape.
CSAVEM	Saves a machine language program on cassette tape.
DATA	Lists constant data created within the program but unaddressable by the BASIC program. It must be stored within an array if manipulation is to be performed; the results of the manipulation are not permanent. Data in DATA lines are accessed by a forward moving (only) READ statement.
EOF	Tests for End of File.
ERL	Gets line number where an error occurred.
ERR	Gets the number of the error which occurred.
ERROR	Simulates an error.
INP	Reads input from a port.
INPUT #	Reads input from a file.
INPUT$	Reads fixed number of characters from keyboard or file.
INST	Searches for a specific string within another string.
LINE INPUT	Inputs a string from the keyboard without regard for punctuation.
LOADM	Loads a machine language program.
LPRINT USING	Prints formatted data on the hardcopy printer.
LPRINT # USING	Writes formatted data to a file.
MOTOR	Turns the cassette motor on and off.
OPEN	Opens a file for input, output, or appending.
OUT	Outputs a byte to a CPU port.
PEEK	Obtains a value directly from memory.

POKE	Stores a value directly to memory.
PRINT #	Writes sequential output to a file.
READ	Accesses internally stored DATA lines and retrieves the data as designated variables, customarily array elements.
RESTORE	Resets the pointer to the first DATA line contained in the program and allows DATA lines to be reread.
SOUND	Outputs a tone, used to play music or alert.

STRING FUNCTIONS

DATE$	Holds the current date.
DAY$	Holds the current day of the week.
INKEY$	Used for a one-character strobe of the keyboard.
INPUT$	Reads fixed number of characters from keyboard or file.
LEN	Returns the length of a string variable.
LEFT$	Allows access to characters on the left side of a string.
LINE INPUT	Input a string from the keyboard without regard for punctuation.
MID$	Allows access to characters in the middle of a string. Replaces middle characters of a string.
RIGHT$	Allows access to characters on the right side of a string.
SPACE$	An expandable constant that allows specification of a string of blank characters at execution time.
STRING$	Defines a string of characters whose maximum length is 255 characters.
STR$	Used to convert a numeric value to a string variable.
TIME$	Gets the current time.

If you study this chapter, you should have a solid grasp of a few important programming techniques, and by and large the majority of the BASIC instructions used on the Model 100, the MEW.

Of course, these instructions are not the sum total of the instructions. Several more fall into two specific categories: scientific and graphics. In the next chapter, we will discuss the features of the scientific in-

struction set, though little time will be spent to build programs using them. Most MEWs will be purchased by the business user, and this book has been designed for that user. Graphics, however, is relevant to the business user, and will be discussed in Chapter 7.

And A Little Bit of Trig

If you've taken trigonometry, then you're familiar with such functions as SINe and COSine, which return, respectively, the sine and cosine of an angle. Frequently, it is important to know the sine of an angle given in radians (as measured by a protractor). If you were to type PRINT SIN(2), you would receive in return the value .909297. However, if your protractor is measured in degrees, it's necessary to convert the degree measurements to the radian measurements useful in trigonometric functions.

For example, let's say you measured an angle and found it to be 30 degrees. To convert that to radians, divide the degree measurement by 57.2958:

```
PRINT SIN(30/57.2958)
```

The answer which will be returned will be .5, the sine of 30 degrees.

It is necessary to divide the degrees by 57.2958 to get the radians, because a semi-circle measures 180 degrees and also measures pi radians. Thus, pi radians = 180 degrees, using pi at a value of 3.14159. Therefore, one radian = 180/3.14159 degrees.

And if you have to calculate it very much, here is a little program to do the job:

```
10 CLS
20 PRINT "COSINE CALCULATOR"
30 INPUT "ANGLE IN DEGREES: ";A
40 PRINT "COSINE = ";COS(A/57.2958)
```

PART 3

MATH AND GRAPHICS

CHAPTER 7

Screen Play

Mathematics

Did you ever notice that every time you open up a programming book the author quickly bypasses the instructions designed for mathematical functions? There's a reason for that. Those authors, myself included, are more oriented to business than they are to the world of mathematics, which should once and for all time dispell the terrible rumor that one must be mathematically inclined to be successful with computers. Hardly.

I'm not a mathematician and this will not be a mathematical treatise. But we will provide demonstrations and build mathematical formulae around them, and then solve them using the MEW. Once you see how the scientific instruction set works, then you can go off on your own TANgent (sorry about that). In the early days of computers, the only way the sign of a number could be tested was to check its relationship to zero. Nowadays it's a little more simple, because we have the SGN function which returns the algebraic sign. In this instance, it is not necessary to know the degree of "minusness" to know that a number is negative. The same is true on the positive side of the scale. The degree of positiveness is not important. The function, then will return a -1 for a negative number, a 0 for the value zero, and a $+1$ (without the $+$ sign) for a positive number. Here's an example:

```
10 A = -50
20 B = 0
30 C = 100
```

```
40 PRINT SGN(A)
50 PRINT SGN(B)
60 PRINT SGN(C)
```

This will return the values −1, 0, and 1 in succession.

Computers have brought another blessing to we who had to learn to obtain the square root of a number the long, hard, tedious way. It is the square root function (SQR). The square of a number is produced by multiplying that number by itself. The number 4 is the square of the number 2. Two, then, is a square root of the number 4. So, if your PRINT SQR(4), the answer will come back as 2. The square root of 2, which seems to be the backbone of much mathematical calculation, is 1.414213562373. My MEW told me.

Trigonometry Begins with a Radian

In mathematics, it is often convenient to use a unit of angular measure called a *radian*. In simple terms, a radian is a measure on the circumference of a circle (call it *s*) which is equal to the radius (call it *r*) of the circle. The number of radians in the angle equals *s*/*r*. One radian is equal to 180 degrees divided by pi, which is equal to 57 degrees, 17 minutes, and 44.8 seconds. One degree is equal to pi divided by 180 degrees, or 0.0174533 radians. Trust me. I looked it up.

Radian is a measure of angle. Connect both ends of that line on the circumference to the center point of the circle, and the formed angle is said to have n radians.

Webster defines tangent in these words, "The ordinate of any point on the terminal side of an angle divided by the nonzero abcissa of this point with the vertex coinciding with the origin of a planned rectangular coordinate system and the initial side of the angle coinciding with the positive X Axis." See why business writers avoid scientific instructions?

The Infernal Triangle

Some definitions are in order. From the triangle in Figure 7-1, these ratios exist:

```
sine a    = y/r
cosine a  = x/r
tangent a = y/x
```

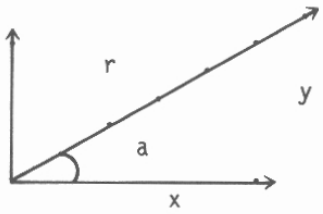


Figure 7-1

COSine at the Bat

Slammin' Sammy Sine steps up to the plate. The pitcher winds up and hurls a burning fastball in the strike zone. Sammy rears back and hits the ball with an initial velocity V (in feet per second) at an angle A to the ground. The height H (in feet) of the ball above the ground is given in terms of the distance X (in feet) from home plate by this equation:

$$H = \left(\frac{16.1X^2}{V^2 \cos^2 A} \right) + X \tan A$$

Now, if one degree is 0.0174533 radians, then 45 degrees is calculated at .7853985 radians. Trust me. I calculated it on my MEW. Sammy hits the ball straight down the first base line, at a 45 degree angle to the ground at a velocity of 40 feet per second. We need to know the height of the ball above the ground when it crosses over first base, which is 90 feet from home plate. Our formula now looks like this:

$$H = \left(\frac{16.1 * 90^2}{40^2 \cos^2 (45 * 0.0174533)} \right) + 90 \tan (45 * 0.0174533)$$

That will produce a program which looks like this:

```
10 A = 45
20 V = 40
30 X = 90
40 R = A * 0.0174533
50 N = (16.1 * (X * X))
```

```
60 D = ((V * V) * (COS(R) * COS(R)))
70 Z = (X * TAN(R))
80 H = (N/D) + Z
90 PRINT H
```

If you run the program, you'll find that the height of Sammy's hit above first base is 253.01267032939 feet. But of course, precision isn't most important, so if you wish to change statement 90 to PRINT INT(H), Sammy won't mind. And if you wish to experiment with the variables, change statements 10–30 to INPUT statements wherein you will gather the values of A, V, and X. The Cosine function (COS) returns the cosine of angle given by the numeric expression. The angle must be given in radians. The Tangent function (TAN) returns the tangent of the angle given by the numeric expression. The angle must also be given in radians; you now know how to determine those. Perhaps when we have finished this chapter, you can use this formula to graph Sammy's hit on the screen—but before you could do that, you must learn more math.

Sound of the ArcTANgent

Look at the first equation on page 195. If the expression tan A were set equal to another factor, i.e., Y = tan A, that same expression could be expressed by A = arc tan Y, which, as can be seen, is another way to determine the radians, or, if you will, the *inverse* of the tangent. The Arctangent function (ATN), then, returns the arctangent of numeric expressions in radians. The resulting value ranges from minus pi to plus pi. (The value of pi is 3.14159265.)

It's No SINe

Recall that the ratio of the side opposite the angle to the radius was determined as the sine. The Law of Sines says that in any triangle, the ratio of the length of a side to the sine of the angle opposite that side is the same for each side-angle pair. The trigonometric sine function is simply: SIN (numeric expression). In the triangle, the function would have been SIN(Y/R).

And the Last Two

The final two scientific instructions are the EXPonential function and the LOGarithmic functions.

Consider the equation:

$$y = b^x$$

In this expression, x is the exponent of b. The expression itself is called the logarithm of y to the base b and is written:

$$x = \text{Log}_b y$$

The EXP function returns the exponential or natural antilog of a numeric expression. It must be in the range of ± 87.3365. The LOG function returns the natural logarithm (base e) of a numeric expression greater than zero.

This program will demonstrate the values:

```
10 FOR N = 1 TO 10
20     PRINT N;LOG(N);EXP(N)
30 NEXT N
```

Derived Functions

To avoid boggling everybody's mind and to avoid a more complete demonstration of just how little trigonometry the author really knows, no explanation of the derived functions will appear here, simply the list which appears in the book. These, like all the other mathematical functions, are already known to the mathematician who needs to know them. If you don't already know what they mean, you probably won't need to.

SECANT
COSECANT
COTANGENT
INVERSE SINE
INVERSE COSINE
INVERSE SECANT
INVERSE COSECANT
INVERSE COTANGENT
HYPERBOLIC COTANGENT
INVERSE HYPERBOLIC SINE
INVERSE HYPERBOLIC COSINE
INVERSE HYPERBOLIC TANGENT
INVERSE HYPERBOLIC SECANT

HYPERBOLIC SINE
HYPERBOLIC COSINE
HYPERBOLIC TANGENT
HYPERBOLIC SECANT
HYPERBOLIC COSECANT
INVERSE HYPERBOLIC COSECANT
INVERSE HYPERBOLIC COTANGENT
PI

Were the author a mathematician, he could no doubt provide a variety of formulas for plotting circles and ellipses and a variety of other shapes on the screen. If you must know that kind of mathematics, then by all means obtain the necessary texts and apply them to the MEW.

Since this book is intended to be about a business machine, and since much of the graphics used in business include plotting business data, then we should spend some time determining how best to use the machine to plot data which are relevant to the business world.

Naturally, scientific programs are used by scientific people. The author is not one of those people. However, it would be useful to explore how the MEW can assist in some of the elements of statistics.

The problem: Given ten numbers from a sample, find the MEAN, the AVERAGE (the INTEGER portion of the MEAN), the RANGE, and the MOVING AVERAGE.

```
10 CLS
20 DIM A(10)
30 DIM B(10)
40 PRINT "ENTER TEN NUMBERS, EACH FOLLOWED BY <E>"
50 FOR N = 1 TO 10
60     INPUT A(N)
70 NEXT N
```

We begin by establishing two arrays large enough to hold ten numbers each. Actually, statements 20 and 30 were not fully necessary, as there is a default dimension of 10. Next the standard input instruction is used to collect ten numbers in succession. No editing is done on the number. A more sophisticated application might wish to do so.

```
80 '********************
90 '* DEVELOP AVERAGE *
100 '********************
110 FOR N = 1 TO 10
120     B = B + A(N)
130 NEXT N
140 MEAN = B/10
150 AVG = INT(B/10)
```

The first routine develops both the mean and the average. For our purposes here, we have established the average as being the integer portion of the mean.

The loop is exercised ten times and the result in B (not to be confused with Array B) is the sum of all numbers. In statement 140, the arithmetic mean is established. The next instruction takes the integer of the same division. Theoretically, the integer of the mean could have been taken with precisely the same result.

```
160 '*****************
170 '* DEVELOP RANGE *
180 '*****************
190 LOW = 9999
200 HIGH = 1
210 FOR N = 1 TO 10
220     IF A(N) > HIGH THEN HIGH = A(N)
230     IF A(N) < LOW THEN LOW = A(N)
240 NEXT N
```

Determining the range is a little easier. All we want to do is to establish the highest and lowest value in the set of numbers. We begin by setting the low value high, so the incoming data should be lower than that value. Conversely, the high is set at a low value, so the incoming data should be higher than that value. After the first number has been read, HIGH and LOW should contain the same value. In this particular problem, a maximum value of 9999 has been arbitrarily established.

Determining the moving average is a bit more difficult. In this instance, the determination has been made to examine data in sets of three. In ten numbers, there are eight sets of three. These values are then stored in eight successive positions within array B, ready to be read and presented on the screen.

```
250 '************************
260 '* DEVELOP MOVING AVERAGE *
270 '************************
280 X = 1
290 FOR N = 1 TO 8
300     B(X) = A(N) + A(N+1) + A(N+2)
310     B(X) = INT(B(X))
320     X = X + 1
330 NEXT N
```

And finally, the presentation of the information on the screen:

```
340 '************************
350 '* PRESENTATION OF DATA *
360 '************************
370 PRINT "MEAN:        ";MEAN
380 PRINT "AVERAGE:     ";AVG
390 PRINT "HIGH:        ";HIGH
400 PRINT "LOW:         ";LOW
410 PRINT "MOVING AVG:  ":
420 FOR N = 1 TO 8
430     PRINT B(N);" ";
440 NEXT N
```

Add this technique to weighted moving averages, standard deviations, and other formulae and you'll be able so see how the program may be applied to the solution of mathematical problems. Here is the program in its entirety:

```
10 CLS
20 DIM A(10)
30 DIM B(10)
40 PRINT "ENTER TEN NUMBERS, EACH FOLLOWED BY <E>"
50 FOR N = 1 TO 10
60     INPUT A(N)
70 NEXT N
80 '********************
90 '* DEVELOP AVERAGE *
100 '********************
110 FOR N = 1 TO 10
120     B = B + A(N)
130 NEXT N
140 MEAN = B/10
150 AVG = INT(B/10)
160 '******************
170 '* DEVELOP RANGE *
180 '******************
190 LOW = 9999
200 HIGH = 1
210 FOR N = 1 TO 10
220     IF A(N) > HIGH THEN HIGH = A(N)
230     IF A(N) < LOW THEN LOW = A(N)
240 NEXT N
250 '**************************
260 '* DEVELOP MOVING AVERAGE *
270 '**************************
280 X = 1
290 FOR N = 1 TO 8
300     B(X) = A(N) + A(N+1) + A(N+2)
310     B(X) = INT(B(X))
320     X = X + 1
330 NEXT N
340 '************************
350 '* PRESENTATION OF DATA *
360 '************************
370 PRINT "MEAN:        ";MEAN
380 PRINT "AVERAGE:     ";AVG
```

```
390 PRINT "HIGH:        ";HIGH
400 PRINT "LOW:         ";LOW
410 PRINT "MOVING AVG: ";
420 FOR N = 1 TO 8
430     PRINT B(N);" ";
440 NEXT N
```

Calculating Compound Interest—Math of Finance

In problems involving compound interest, simple interest is calculated over the principal during some interval and is then added to the principal before the next calculation is made. If P is the principal, i is the interest rate (as a decimal), and n the number of years involved, then the basic formula for the amount after n years is:

$$A = P(1 + i)^n$$

What it means is this: if you have \$1,000 (P) to be invested for 10% (i) for 5 years (n), the calculation would work out like this:

$A = P(1 + i)^n$
$A = 1000(1 + .1)^5$
$A = 1000(1.1)^5$

```
      1.1      (1)
  ×   1.1      (2)
    1.21
  ×   1.1      (3)
   1.331
  ×   1.1      (4)
  1.4641
  ×   1.1      (5)
 1.61051
```

Therefore, $A = 1000(1.61061) = \$1{,}610.51$.

You can prove that to yourself with the program which follows. The "n" has been changed to "Y."

```
10 CLS
20 INPUT "ENTER PRINCIPAL: ";P
30 INPUT "ENTER INTEREST:  ";I
40 INPUT "ENTER YEARS:     ";Y
50 TV = (1+I)
60 FOR N = 1 TO Y-1
70     TV = (TV * (1+I))
80 NEXT N
90 PRINT "PAYOFF IS: ";(P*TV)
```

Enter this program and the answer you get should be similar to that which was worked out by hand.

Occasionally, we tend to make a program much more complicated than it should be. The following is a program whose function it is to calculate factorials. A *factorial,* you will recall, is a number multiplied by itself in a diminishing progression. Factorials are useful for combinatorial and permutational calculations. This program will handle a double precision number up to a value of 32. Since that returns an answer of 8.683317618811887D+36, there shouldn't be any major difficulty.

```
10 CLS
20 DEFDBL A,B
30 PRINT "THIS PROGRAM CALCULATES FACTORIALS"
40 PRINT
50 INPUT "ENTER THE NUMBER : ";A
60 PRINT
70 B = A
80 PRINT A;"x";
90 FOR N = A-1 TO 2 STEP -1
100     B = B * N
110     PRINT N;"x";
120 NEXT N
130 PRINT " 1"
140 PRINT
150 PRINT "THE ANSWER IS: ";B
```

Give this program the number 8!, for example, and the display will show 8 × 7 × 6 × 5 × 4 × 3 × 2 × 1, with a final value of 40320. Other numbers will produce similar results.

CHAPTER 8

Graphics

While there are eight lines of 40 characters for printed text, graphics works somewhat differently. Look carefully at any character and you will see that each character is made up of cells or elements. Actually, they are called "picture elements," or *pixels*, for short. Each character is made up of a 6-wide by 8-high pixel matrix. Of that 6 by 8 matrix, 5 (wide) and 7 (high) are used to form the character, Figure 8-1 shows the letter E:

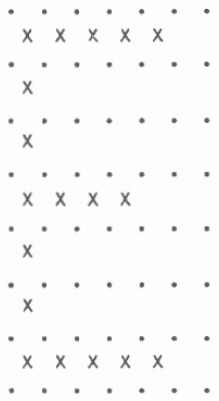


Figure 8-1

The pixels are individually addressable. There are 240 pixels (numbered 0 to 239) on the X-axis, i.e., the length of the screen. There are 64 pixels (numbered 0 to 63) on the Y-axis, i.e., the height of the screen.

This program will demonstrate that by first filling the screen vertically, left to right, and then erasing it horizontally, top to bottom, the program works at the pixel level within the nested FOR. . .NEXT loops. Enter and run it:

```
10 CLS
20 FOR X = 0 TO 239
30     FOR Y = 0 TO 63
40         PSET (X,Y)
50     NEXT Y
60 NEXT X
70 FOR Y = 0 TO 63
80     FOR X = 0 TO 239
90         PRESET (X,Y)
100    NEXT X
110 NEXT Y
120 MENU
```

In this example, the first of the two graphics commands are introduced: PSET and PRESET. PSET turns the pixel on. PRESET turns it off. Formats of the two instructions are the same:

```
PSET (x,y)
PRESET (x,y)
```

The instruction inspects the X-axis value first; it must be in the range of 0 to 239. It inspects the Y-axis value next; it must be in the range of 0 to 63. You may specify an actual decimal value for one or both operands, as well as the derived value which was used in the program. Note that in the program, all pixels in a character matrix were used.

Next, the MEW has a plethora of graphics characters, as can be seen from the table in the appendix of your Model 100 Reference Manual. These occupy ASCII references from 123 to 255. They may be entered manually using the CODE and GRPH keys, or they may simply be printed on the screen. Enter and run this program to get an idea of how they will appear:

```
10 CLS
20 FOR N = 123 TO 255
30     PRINT N;CHR$(N)
40     PRINT
50     FOR Z = 1 TO 500:NEXT Z
60 NEXT N
```

These are set to move slowly so you may see the nature of the characters being printed. If you would like it to move faster, reduce the 500 in statement 50.

Graphics may be sedentary, such as this histogram framework:

```
10 CLS
20 FOR X = 51 TO 185 STEP 12
30     LINE (X,10)-(X+8,40),1,B
40 NEXT X
50 PRINT @249,"J F M A M J J A S O N D"
60 LINE (40,40)-(200,40)
70 LINE (42,10)-(42,40)
80 FOR Y = 10 TO 40 STEP 5
90     LINE (41,Y)-(43,Y)
100 NEXT Y
500 GOTO 500
```

Or they may be dramatic. Here is a thunderbolt:

```
10 CLS
20 LINE (20,20)-(150,60)
30 LINE (20,20)-(150,60),0
40 GOTO 40
```

Both illustrations use the LINE command, whose purpose is, of course, to draw a line from one point (known as x1,y1 coordinates) to a second point (x2,y2 coordinates). In line 20 of the thunderbolt example, that's precisely what happens. The same is true in lines 60, 70, and 90 of the histogram example, though the coordinates developed there use numeric variables instead of the actual numbers.

The format of the LINE instruction is:

LINE (x1,y1)–(x2,y2),switch,BF

The coordinates are actual, specific points on the LCD screen. Coordinate x must lie between 0 and 239. Coordinate y must be between 0 and 63. The dash between the coordinates is mandatory. The *switch* operand contains a number. If it is an even number (0, 2, 4, etc.), the line is drawn. If it is an odd number (1, 3, 5, etc.), the line is erased. Actually, to be precise, the pixels on that line are drawn or erased. The switch operand need only be specified if one of the remaining two operands, B or F, has been specified.

When operand B has been specified, you will be "boxing in" the figure whose coordinates you have specified; thus, you must specify opposite corners of a square or rectangle. Sorry, it is not possible, except by extensive planning, to develop other shapes. The operand F is the "fill" command. When used, it will darken the inside of the box which has been specified. It is possible to fill with no color just as it is possible to fill with color, as per these instructions:

```
10 CLS
20 LINE (50,40)-(90,10),1,BF
30 LINE (60,30)-(80,20),0,BF
```

In this example, we have simply drawn a box, filled it with color, and then drawn a second box within the first and filled it with no color.

Occasionally you must determine, through the program, just where the cursor is on the screen. The reason is that you may have developed some graphics figures which have been based upon a formula whose variables are not known when the program was written. Location of the cursor on the screen happens by using these two instructions:

CSRLIN returns the cursor line (Y-axis) where the cursor is located when the instruction was executed. Valid numbers will lie in the range of 0 to 7. Format is:

```
VP = CSRLIN
```

POS returns the horizontal screen position (X-axis) of the cursor. Valid numbers lie in the range of 0 to 39. This instruction requires a dummy operand, like the FRE instruction. Format is:

```
100 HP = POS(0)
```

The same function occurs for the printer head within the printer buffer with the instruction LPOS. Its format is:

```
110 PP = LPOS(0)
```

Let's return to the histogram. That program is duplicated here:

```
10 CLS
20 FOR X = 51 TO 185 STEP 12
30     LINE (X,10)-(X+8,40),1,B
40 NEXT X
50 PRINT @249,"J F M A M J J A S O N D"
60 LINE (40,40)-(200,40)
70 LINE (42,10)-(42,40)
80 FOR Y = 10 TO 40 STEP 5
90     LINE (41,Y)-(43,Y)
100 NEXT Y
500 GOTO 500
```

Statement 500 is a GOTO itself, placed far enough ahead so it will not be overwritten. It freezes the display, else a scroll would occur, destroying the graphics. Statements 10 through 40 clear the screen and

draw 12 vertically oriented rectangles. Statement 50 provides a legend for each month of the year. Statements 60 and 70 draw the axis marks. And statements 80 to 100 draw hash marks along the left axis at increments of five. There are six increments of five. For our discussion, we will use but five of the six; each of the five represents 20 percent of the scale.

Add these instructions:

```
110 PRINT @42,"Each";
120 PRINT @82,"Mark";
130 PRINT @122," Is";
140 PRINT @162,"20 %";
```

This provides some legend to the scale marked on the Y-axis. The purpose of the trailing semicolon is to avoid the scroll.

Next, the scale of the data to be used must be determined. There are two ways to do that—determine the range of the data by first reading all the data or determine what is the acceptable range of data. We will take the latter approach. Add these instructions:

```
150 DATA 10,38,13,42
160 DATA 65,17,72,89
170 DATA 44,55,20,47
```

These are the sales figures on the sale of Widgets by Consolidated Gizmo for the last fiscal year. Note that the DATA lines can be placed anywhere in a program, even in the line of executing instructions. We'll begin by loading them to an array, as we will need the data a second time:

```
180 DIM FY(12)
190 RESTORE
200 FOR N = 1 TO 12
210     READ FY(N)
220 NEXT N
```

Now the figures are loaded into the array, and we see how nicely they fit in a scale of 0–100. Now we have a problem. Those hash marks are set off at a scale of five pixels. We don't have 100 pixels available, just 25. This means that our chart won't be precise. We'd better modify the data. To round the data up or down, determine the number of times it can be divided by 5. If there is an equal division (the INTeger of the dividend is identical to the dividend itself), then we have no problem. If the division is not equal, then the position of that inequality relative to the midpoint (5) must be determined. Enter these instructions:

```
230 FOR N = 1 TO 12
240 IF INT(FY(N)/5) = (FY(N)/5) THEN 290
250 T = INT(FY(N)/5)
260 IF (FY(N)/5) - T < 0.5 THEN 290
270 FY(N) = INT(FY(N)) + 1
280 GOTO 300
290 FY(N) = INT(FY(N)
300 NEXT N
```

At this point the data have been adjusted and reside in the array. To determine the relationship of that data to the zero scale on the graph, draw a new box for it, and fill it in. Go back and study instruction 30. Each of the 12 rectangles is drawn from position X, beginning at 51 and proceeding to 185 in steps of 12. The height of the rectangle lies in the Y-axis positions 10 to 40, or 30 positions. Of those 30 positions, we intend to use 25, specifically 15 to 40. Now, this means that these positions correspond to these percentages:

POSITIONS		PERCENTAGES
15	–	100
20	–	80
25	–	60
30	–	40
35	–	20
40	–	0

As can be seen, there is an inverse relationship. The trick is to locate the data on the 15 to 40 scale in the same ratio as the 100 to 0 scale.

Recall that we scaled the data on the basis of the relationship to 5. That means that there are 25 elements (that number will be derived by dividing the matrix number by 4) to be plotted. But the 25 elements must be measured from zero, or from pixel position 40. So, at all times, we'll be dealing with the number of positions away from 40 (less than 40) plus 1. For example, if the number we wish to plot is 17, then (40 − 17 + 1) will tell us to draw the top side of our new box. Here we go:

```
310 N = 1
320 FOR X = 51 TO 185 STEP 12
330     Y = INT(FY(N)/4):N = N + 1
340     LINE (X,Y)-(X+8,40),1,BF
350 NEXT X
```

That's how histograms are made. Here is the program in its entirety:

```
10 CLS
20 FOR X = 51 TO 185 STEP 12
30     LINE (X,10)-(X+8,40),1,B
40 NEXT X
50 PRINT @249,"J F M A M J J A S O N D"
60 LINE (40,40)-(200,40)
70 LINE (42,10)-(42,40)
80 FOR Y = 10 TO 40 STEP 5
90     LINE (41,Y)-(43,Y)
100 NEXT Y
110 PRINT @42,"Each";
120 PRINT @82,"Mark";
130 PRINT @122," Is";
140 PRINT @162,"20 %";
150 DATA 10,38,13,42
160 DATA 65,17,72,89
170 DATA 44,55,20,47
180 DIM FY(12)
190 RESTORE
200 FOR N = 1 TO 12
210     READ FY(N)
220 NEXT N
230 FOR N = 1 TO 12
240 IF INT(FY(N)/5) = (FY(N)/5) THEN 290
250 T = INT(FY(N)/5)
260 IF (FY(N)/5) - T < 0.5 THEN 290
270 FY(N) = INT(FY(N)) + 1
280 GOTO 300
290 FY(N) = INT(FY(N)
300 NEXT N
310 N = 1
320 FOR X = 51 TO 185 STEP 12
330     Y = INT(FY(N)/4):N = N + 1
340     LINE (X,Y)-(X+8,40),1,BF
350 NEXT X
500 GOTO 500
```

Graphics in 3-D

The presentation of graphics in three dimensions isn't easy, but it isn't overly difficult either. It does, however, take some advance planning. Let's take the histogram problem one step further, and show how we might approach the presentation of three year's worth of data. The illustration we're about to build will show the columns only. We'll leave it to your ingenuity to adapt the data to it. We begin by laying out the same 12 rectangular blocks we used with the histogram. Enter and run the program this far:

```
10 CLS
20 FOR X = 50 TO 205 STEP 15
30     LINE (X,10)-(X+8,40),1,B
40     LINE (X,9)-(X+2,7)
50     LINE (X+8,9)-(X+10,7)
60     LINE (X+9,39)-(X+11,37)
70     LINE (X+11,7)-(X+11,37)
80     LINE (X+2,7)-(X+10,7)
90 NEXT X
```

Note that in statement 20 we began the display one pixel earlier and spread the display wider on the screen. Statement 30 defined the basic box, and the B operand drew a line around it. There is no fill at this time. Statements 40 through 80 build the third dimension, and are a function of planning. That aspect will be discussed further into the text.

The second set of rectangles are dropped five pixels down and five pixels to the left. The drawing of these rectangles is the same, with the exception that the box which is drawn is filled with color. The three dimensional aspects are different, and will require some additional work.

```
100 FOR X = 45 TO 200 STEP 15
110     LINE (X,15)-(X+8,45),1,BF
120     LINE (X,14)-(X+2,12)
130     LINE (X+8,14)-(X+10,12)
140     LINE (X+9,44)-(X+11,42)
150     LINE (X+11,12)-(X+11,42)
160     LINE (X+2,12)-(X+10,12)
170 NEXT X
```

Enter the program statements to this point and run them. Note that the third dimension of this second set of rectangles is cluttered by the lines of the first set of rectangles. Unfortunately, the colorfill features of this system do not work within any defined border. They work only within the rectangle itself. Thus, the two sides which comprise the third dimension cannot be easily filled with color or no color, necessitating the coding which will clear out the extraneous lines:

```
180 FOR X = 45 TO 200 STEP 15
190     LINE (X+2,13)-(X+8,13),0
200     LINE (X+1,14)-(X+7,14),0
210     LINE (X+9,14)-(X+9,43),0
220     LINE (X+10,13)-(X+10,42),0
230 NEXT X
```

The third set of rectangles covers in exactly the same ratio, except that the fill is now no color, and that portion of the second set of rectangles is now removed. Unfortunately, in the removal, the box for the

third rectangle is also removed, necessitating the use of statement 260 to redraw the box.

```
240 FOR X = 40 TO 195 STEP 15
250     LINE (X,20)-(X+8,50),0,BF
260     LINE (X,20)-(X+8,50),1,B
270     LINE (X,19)-(X+2,17)
280     LINE (X+8,19)-(X+10,17)
290     LINE (X+9,49)-(X+11,47)
300     LINE (X+11,17)-(X+11,47)
310     LINE (X+2,17)-(X+10,17)
320 NEXT X
```

And the final routine cleans up the third dimension for this set of rectangles:

```
330 FOR X = 40 TO 195 STEP 15
340     LINE (X+2,18)-(X+8,18),0
350     LINE (X+1,19)-(X+7,19),0
360     LINE (X+9,19)-(X+9,48),0
370     LINE (X+10,18)-(X+10,47),0
380 NEXT X
```

At the moment, it looks as if three dozen milk cartons are sitting in perfect formation. In the same manner as data were adapted to the histogram, data must be adapted to these columns. The height will be varied. However, bear in mind that you are dealing with the third dimension, as well as just the two dimensions of the histogram.

While the emphasis has been on the building of a three-dimensional histogram, this technique has several other uses. Certainly it can be used to show the data which have accumulated in tables. But it can also show the data which have accumulated in tables. But it can also be used to show relief, such as would be seen if the skyline of a city were to be depicted. Naturally you wouldn't want to spend much time depicting the skyline of a city on a computer this size, but a draftsman might want to experiment with a mechanical drawing. Bear in mind that this computer is intended to be an accessory, not a computing workhorse.

One danger is that this machine might be applied to tasks for which a larger machine would be more useful. However, if your business involves sales, the ability to bring out the MEW and to graph your client's data might be the clincher necessary to gain the sale.

Here is the program in its entirety:

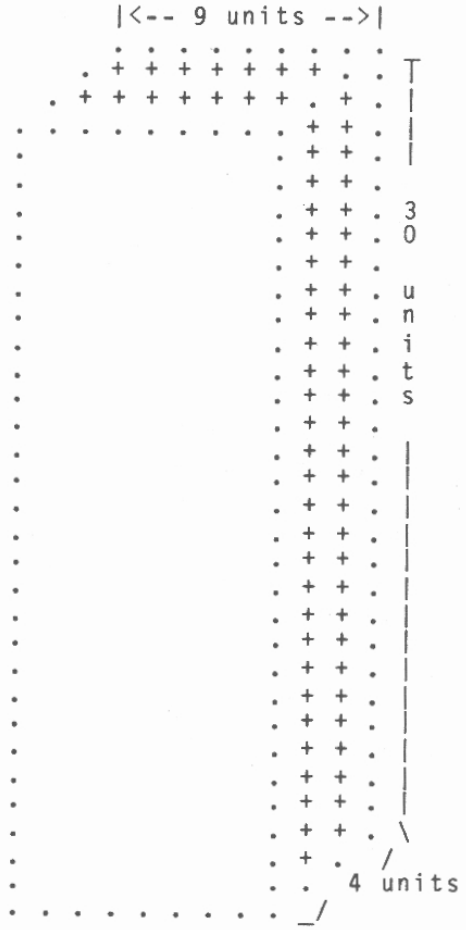
```
10 CLS
20 FOR X = 51 TO 205 STEP 15
30     LINE (X,10)-(X+8,40),1,B
40     LINE (X,9)-(X+2,7)
```

```
50      LINE (X+8,9)-(X+10,7)
60      LINE (X+9,39)-(X+11,37)
70      LINE (X+11,7)-(X+11,37)
80      LINE (X+2,7)-(X+10,7)
90 NEXT X
100 FOR X = 45 TO 200 STEP 15
110     LINE (X,15)-(X+8,45),1,BF
120     LINE (X,14)-(X+2,12)
130     LINE (X+8,14)-(X+10,12)
140     LINE (X+9,44)-(X+11,42)
150     LINE (X+11,12)-(X+11,42)
160     LINE (X+2,12)-(X+10,12)
170 NEXT X
180 FOR X = 45 TO 200 STEP 15
190     LINE (X+2,13)-(X+8,13),0
200     LINE (X+1,14)-(X+7,14),0
210     LINE (X+9,14)-(X+9,43),0
220     LINE (X+10,13)-(X+10,42),0
230 NEXT X
240 FOR X = 40 TO 195 STEP 15
250     LINE (X,20)-(X+8,50),0,BF
260     LINE (X,20)-(X+8,50),1,B
270     LINE (X,19)-(X+2,17)
280     LINE (X+8,19)-(X+10,17)
290     LINE (X+9,49)-(X+11,47)
300     LINE (X+11,17)-(X+11,47)
310     LINE (X+2,17)-(X+10,17)
320 NEXT X
330 FOR X = 40 TO 195 STEP 15
340     LINE (X+2,18)-(X+8,18),0
350     LINE (X+1,19)-(X+7,19),0
360     LINE (X+9,19)-(X+9,48),0
370     LINE (X+10,18)-(X+10,47),0
380 NEXT X
```

What we've just done is to draw a picture which was derived by experimentation. That will always be the case—you will derive complex figures by experimentation. From that experimentation, you will draw a generality, and then build a formula on the basis of that generality. For instance, observe Figure 8–2. This illustration gives some basic idea of the relationships. First, the figure takes up 9 units, or points on the X-axis. Therefore, no matter where the figure is to be placed relative to the 240 X-axis pixels, once the beginning point has been established, X will be that point, and X+8 will be the other X-axis point.

Next, the figure is 30 units high. This is, of course, the size of the scale of the figure. Anything less than the scale of the figure must be marked accordingly. Recall that you are really talking about drawing your figure from the position of (Y + 29 − value + 1), assuming that Y is the topmost point of the scale.

The third dimension is constructed at a depth of four units, set at a 45 degree angle. Once you have your corner points (X, X + 8, and Y + 29 units), you may calculate the location of the bounds of the third



dimension (X + 3, X + 11, and the two Y coordinates, Y − 3 and Y + 26).

The surface area of the third dimension is changed in relation to the lines which connect the three dimensions. On top, the first line has the coordinates (X + 1, X + 7, Y − 1, and Y − 2). On the side, the first line has the coordinates (X + 9, X + 10, Y − 1, and Y − 2).

Thus, the figure may be derived mathematically, once the relationships have been determined. Whereas we made this program work, the formula for the figure may be developed and applied universally wherever such a figure is required.

Line Graphs

We should not overlook the potential for plotting the data simply with a line. The following program takes the data devised previously as

monthly sales figures and lays it out in line graph form against the same scale:

```
10 CLS
20 RESTORE
30 DIM P(12)
40 DATA 56,68,80,92
50 DATA 104,116,128,140
60 DATA 150,164,176,189
70 FOR N = 1 TO 12
80     READ P(N)
90 NEXT N
```

In these opening instructions, we have loaded the X-axis position (pixel position) for each of the 12 elements of the graph. They were changed to center immediately above the letter of the month. Next comes the loading of the matrix with the data which represent the sales by month. These data could be entered from the keyboard instead of being placed in DATA lines.

```
100 DIM D(12)
110 DATA 10,38,13,42
190 DATA 65,17,72,89
120 DATA 44,55,20,47
130 FOR N = 1 TO 12
140     READ D(N)
150 NEXT N
```

Now the drawing of the axes and the apportionment of scale:

```
160 PRINT @249,"J F M A M J J A S O N D"
170 LINE (40,40)-(200,40)
180 LINE (42,10)-(42,40)
190 FOR Y = 10 TO 40 STEP 5
200     LINE (41,Y)-(43,Y)
210 NEXT Y
220 PRINT @42,"Each";
230 PRINT @82,"Mark";
240 PRINT @122," Is";
250 PRINT @162,"20 %";
```

The next step is a little less sophisticated than the previous step. The purpose is to create numbers (integers) in the range of the graph. For this discussion, no attempt has been made to round up or down on the basis of the midpoint (that has already been discussed). Thus the data that looked like this:

10, 38, 13, 42, 65, 17, 72, 89, 44, 55, 20, 47

now looks like this:

2, 7, 2, 8, 13, 3, 14, 17, 8, 11, 4, 9

With rounding, for example, the second number would have become 8. This is the coding which accomplishes this:

```
260 FOR N = 1 TO 12
270     D(N) = INT(D(N)/5)
280 NEXT N
```

Now comes the interesting part. With 12 numbers, we have 11 pairs: 1-2, 2-3, etc. These pairs correspond to ends of a line to be drawn. Thus, the first of the pair is located at the current location of X. The second of the pair is located at the offset location of X. The value of each is located at a corresponding value of Y. This is the coding:

```
290 FOR X = 1 TO 11
300     LINE (P(X),(41-(D(X))))-(P(X+1),(41-(D(X+1))))
310 NEXT X
320 GOTO 320
```

Don't let statement 300 frighten you. It is lengthy and seems complex. In fact, it will take up two lines on your screen. But at the same time, note that the number of left and right parentheses balances. For every left parenthesis, you must have a right parenthesis. Recall that the format of the LINE instruction is:

LINE (x1,y1) – (x2,y2)

So you have two pairs of parentheses already—one which contains the *from* address of the line and the other which contains the *to* address; each address is an x,y coordinate. Now, let's break down the instruction:

```
300     LINE (P(X),(41-(D(X))))-(P(X+1),(41-(D(X+1))))

          x1 = P(X)                        x2 = P(X+1)
  from:                              to:
          y1 = (41-(D(X)))                 y2 = (41-(D(X+1)))
```

X, of course, is the number of the iteration of the loop in which this instruction has been placed. Thus, X and X+1 form the pair of data at the end of each line. X and X+1 are the positions of the data drawn from the Position array (P). Next, the bottom position of the display is

on line 40. Recall from previous discussion that the location above the line would be the line (40) less the value plus one. To shorten the process, we have indicated 41 as the location of the line, or 40 + 1. The values located at positions X and X+1 in the Data array (D) are subtracted from 40 to determine the position above the scale line where the data are to be plotted.

That's all there is to it. Study the instruction; enter it and watch it work. Here is the program in its entirety:

```
10 CLS
20 RESTORE
30 DIM P(12)
40 DATA 56,68,80,92
50 DATA 104,116,128,140
60 DATA 150,164,176,189
70 FOR N = 1 TO 12
80     READ P(N)
90 NEXT N
100 DIM D(12)
110 DATA 10,38,13,42
190 DATA 65,17,72,89
120 DATA 44,55,20,47
130 FOR N = 1 TO 12
140     READ D(N)
150 NEXT N
160 PRINT @249,"J F M A M J J A S O N D"
170 LINE (40,40)-(200,40)
180 LINE (42,10)-(42,40)
190 FOR Y = 10 TO 40 STEP 5
200     LINE (41,Y)-(43,Y)
210 NEXT Y
220 PRINT @42,"Each";
230 PRINT @82,"Mark";
240 PRINT @122," Is";
250 PRINT @162,"20 %";
260 FOR N = 1 TO 12
270     D(N) = INT(D(N)/5)
280 NEXT N
290 FOR X = 1 TO 11
300     LINE (P(X),(41-(D(X))))-(P(X+1),(41-(D(X+1))))
310 NEXT X
320 GOTO 320
```

Block Graphics

We should not leave this chapter without some discussion of block graphics, i.e., preparing visual images through the characters available within the character set itself. Recall that these may be invoked using the CHR$ command. You will discover also that they may be invoked in sets using the STRING$ command.

The following program draws a pine tree:

```
10 CLS
20 PRINT SPACE$(3);CHR$(252);CHR$(254)
30 PRINT SPACE$(2);CHR$(252);STRING$(2,239);CHR$(254)
40 PRINT SPACE$(1);CHR$(252);STRING$(4,239);CHR$(254)
50 PRINT CHR$(252);STRING$(6,239);CHR$(254)
60 PRINT SPACE$(3);STRING$(2,239)
```

If you are not in the Yuletide spirit or do not hail from Maine (the Pine Tree state), you'll probably wonder whether you'll ever want to draw a pine tree on your MEW. Probably not, but a number of things are demonstrated by this program. First, note the diagonals, while a perfect 45-degree angle, do not produce a continuous line. The reason for this is that the blocks in which letters are produced are not square. They are taller than they are wide.

Next, note that positioning relative to the left margin may be done using the SPACE$ command. These were printed at the *next* available line, but could have been located via a PRINT @ instruction. And finally, note that the characters may be multiplied using the STRING$ instruction.

The same result may be obtained via the concatenation process, wherein the elements of the graph are simply joined together into a single string, like this:

```
10 CLS
20 PRINT SPACE$(3)+CHR$(252)+CHR$(254)
30 PRINT SPACE$(2)+CHR$(252)+STRING$(2,239)+CHR$(254)
40 PRINT SPACE$(1)+CHR$(252)+STRING$(4,239)+CHR$(254)
50 PRINT CHR$(252)+STRING$(6,239)+CHR$(254)
60 PRINT SPACE$(3)+STRING$(2,239)
```

The next step in the process is known as *String Packing,* a technique whereby graphics may be placed on the screen as rapidly as possible. The concept is to concatenate the string into a string variable, and then print the string variable, like this:

```
10 CLS
20 A$ =  SPACE$(3)+CHR$(252)+CHR$(254)
            +CHR$(10)+CHR$(13)
30 B$ =  SPACE$(2)+CHR$(252)+STRING$(2,239)+CHR$(254)
            +CHR$(10)+CHR$(13)
40 C$ =  SPACE$(1)+CHR$(252)+STRING$(4,239)+CHR$(254)
            +CHR$(10)+CHR$(13)
50 D$ =  CHR$(252)+STRING$(6,239)+CHR$(254)
            +CHR$(10)+CHR$(13)
60 E$ =  SPACE$(3)+STRING$(2,239)
70 PRINT A$;B$;C$;D$;E$
```

What's that? Where did those CHR$(10) and CHR$(13) instructions come from? Glad you asked. To begin with, the CHR$(10) is a linefeed. And the CHR$(13) is a carriage return. Whenever you press the ENTER key, you in effect receive both of these codes appended to the end of what you have entered. If the CHR$(13) is encountered alone, the cursor is returned to the head of the very same line. The use of the CHR$(10) immediately prior to the CHR$(13) causes the linefeed and then the carriage return. Thus, the PRINT instruction of statement 70 can be joined with semicolons for continuous print, with the knowledge that the appropriate linefeed/carriage return characters have been positioned in their proper places.

We're going to take our pine tree one step further and show how the same thing is accomplished using DATA lines. Consider this coding:

```
10 CLS
20 RESTORE
30 FOR N = 1 TO 1000
40     READ X
50     IF X = 9999 THEN 80
60     PRINT CHR$(X);
70 NEXT N
80 END
100 DATA 32,32,32,252,254,10,13
110 DATA 32,32,252,239,239,254,10,13
120 DATA 32,252,239,239,239,239,254,10,13
130 DATA 252,239,239,239,239,239,239,254,10,13
140 DATA 32,32,32,239,239,9999
```

In this instance the graphics routine is built somewhat like the music routine discussed in an earlier chapter. The specific ASCII codes which are to comprise the graphics picture have been placed in sequence in the DATA lines. Note that the values for linefeed (10) and carriage return (13) have been placed there as well.

One instruction doesn't fit into any category:

IPL "filename"

Using this instruction you can designate a program which will run when the system is turned on. IPL is an acronym from *I*nitial *P*rogram *L*oad. That program can perform any function.

Summary

As we end our discussion on the instruction set, here is a summary of the instructions discussed in this chapter:

ATN	Returns the arctangent of a numeric expression in radians. The resulting value ranges from minus to plus pi.
COS	Returns the cosine of the angle given by the numeric expression. The angle must be given in radians.
CRSLIN	Returns the vertical position (line number) of the cursor, in the range of 0 to 7.
EXP	Returns the exponential or natural antilog of the numeric expression. The numeric expression must be in the range of ± 87.3365 or an overflow error will occur. EXP is the opposite of LOG.
IPL	Defines the program to be automatically executed during warm startup.
LINE	Draws the line from the coordinates x1,y1 to x2,y2. The X coordinates are numeric expressions in the range of 0 to 239 while the Y coordinates are numeric expressions in the range of 0 to 63. The instruction has a switch option for illuminating or extinguishing pixels, a code for developing a box, and a code for filling the box with color.
LOG	Returns the natural logarithm of the numeric expression, which must be greater than zero.
LPOS	Returns the current position of the printer head in the printer buffer.
POS	Returns the horizontal screen position of the cursor.
PRESET	Turns off an LCD pixel. X and Y coordinates in the accepted range are required.
PSET	Turns on an LCD pixel. X and Y coordinates in the accepted range are required.
SGN	Returns the algebraic sign of a numeric expression. A −1 is returned for negative numbers, 0 for 0 numbers, and 1 for positive numbers.
SIN	Returns the trigonometric sine of a numeric expression.
SQR	Returns the square root of a numeric expression.
TAN	Returns the trigonometric tangent of a numeric expression.

One Final Word

Now, at the end of this book, we've attempted to deal more with the reason why the MEW works rather than the how it works. It was not

our purpose to provide a book of programs. Others will produce books of programs and a rather substantial market will develop around the machine. There is already a national magazine. Others, such as *80 Microcomputing*, have devoted a section to it, called "C-Notes".

An old proverb says: "Give a man a fish, and he will have food for a day. Teach a man to fish, and he will have food for a lifetime." Nothing so dramatic is promised for the reader of this book, but if knowing what we have tried to teach is important, then this can become a most useful tool.

```
10 CLS
20 RESTORE
30 PRINT @131,"";
40 FOR N = 1 TO 1000
50     READ X
60     IF X = 9999 THEN 90
70     PRINT CHR$(X);
80 NEXT N
90 PRINT @45,CHR$(235)+STRING$(28,231)+CHR$(236);
100 PRINT @205,CHR$(237)+STRING$(28,231)+CHR$(238);
110 FOR N = 85 TO 165 STEP 40
120     PRINT @N,CHR$(233);
130     PRINT @N+29,CHR$(234);
140 NEXT N
150 GOTO 150
160 DATA 84,72,65,84,39,83,32
170 DATA 65,76,76,44,32
180 DATA 70,79,76,75,83,9999
```

APPENDIX A

Instruction Repertoire

ATN

Function: Returns the arctangent of a numeric expression in radians. The resulting value ranges from minus to plus pi.

Form: ATN (numeric expression)

Example:
```
10 A = .5
20 ARC = ATN (A)
```

BEEP

Function: Beeps the speaker—a function also performed by PRINT CHR$(7).

Form: BEEP

Example:
```
10 BEEP
```

CALL

Function: Calls a machine level subroutine.

Form: CALL address, expression 1, expression 2

Example:
```
CALL 42378,10,VARPTR(A$)
```

CDBL

Function: Converts a numeric expression to double precision.

Form: CDBL (numeric expression)

Example:
```
A# = CDBL (A%)
```

CHR$

Function: Displays, on the screen or the line printer, any ASCII character within the range of 0–255. The same ASCII character may generate entirely different display characteristics on the LCD display and the printer.

Form: CHR$ (numeric expression)

Example:
```
10 PRINT CHR$(7)
20 PRINT CHR$(X)
```

CINT

Function: Converts a numeric expression to an integer.

Form: CINT (numeric expression)

Example:
```
A% = CINT(34.56)
```

CLEAR

Function: Clears program variables and dimensioned tables. It is required whenever string variables and arrays occupy more than 256 bytes of memory.

Form: CLEAR string space, high memory

Example:
```
10 CLEAR 100,42000
```

CLOAD

Function: Loads a program from cassette tape. The operand R will cause the program to run upon completion of the load.

Form: CLOAD "filename",R

Example:
```
CLOAD "MEWPRG",R
```

CLOAD?

Function: Loads a program from cassette tape, verifying it against the program stored in memory.

Form: CLOAD? "filename"

Example:
```
CLOAD? "MEWPRG"
```

CLOADM

Function: Loads a machine language program from cassette tape.

Form: CLOADM "filename"

Example:
```
CLOADM "MEWPRG"
```

CLOSE

Function: Closes the files specified. The command assumes that the file is already open and that the numbers in the list correspond to the buffers assigned.

Form: CLOSE file number list
Example: CLOSE 1,2,3

CLS

Function: Clears the screen and puts the cursor in home (upper left) position.
Form: CLS
Example: 10 CLS

COM ON/OFF/STOP

Function: Enables/disables the communications interrupt. If a character is received at the RS-232 port, BASIC jumps to the subroutine defined on the ON COM command. ON enables; OFF disables; STOP disconnects.
Form: COM ON or OFF or STOP
Example:
```
100 COM ON
110 COM OFF
120 COM STOP
```

CONT

Function: Resumes execution of a program after the BREAK key has been pressed or the program has been stopped via the STOP command.
Form: CONT
Example: CONT

COS

Function: Returns the cosine of the angle given by the numeric expression. The angle must be given in radians.
Form: COS (numeric expression)
Example: 10 Y = COS(60 * 0.1745329)

CRSLIN

Function: Returns the vertical position (line number) of the cursor, in the range of 0–7.
Form: CRSLIN
Example: 10 Y = CRSLIN

CSAVE

Function: Saves a program on cassette tape. The A operand will save the file in ASCII format.

Form: CSAVE "filename",A
Example: CSAVE "MEWPRG",A

CSVEM

Function: Saves a machine language program on cassette tape.
Form: CSAVEM "filename",SA, NA, EA
SA = Start address
NA = End address
EA = Entry address
Example: CSAVEM "MEWPRG",40000,48205,40510

CSNG

Function: Converts a numeric expression to single precision.
Form: CSNG (numeric expression)
Example: 10 A! = CSNG(0.66666666666)

DATA

Function: Lists constant data created within the program but unaddressable by the BASIC program. It must be stored within an array if manipulation is to be performed; the results of the manipulation are not permanent. Data in DATA lines are accessed by a forward moving (only) READ statement.
Form: DATA constant list
Example: 100 DATA 20, 24, "M100","Ken Lord"

DATE$

Function: Holds the current date in string form in the MM/DD/YY format, where 01≤MM≤12, 01≤DD≤31, and 00≤YY≤99. BASIC automatically updates the date when the clock rolls through midnight.
Form: DATE$
Example: 10 DATE$ = "07/31/83"
20 PRINT DATE$

DAY$

Function: Holds the current day of the week in string form. Acceptable values are Mon, Tue, Wed, Thu, Fri, Sat, and Sun.
Form: DAY$
Example: 10 DAY$ = "Fri"
20 PRINT DAY$

DEFtype

Function: Defines default variable types where "type" may be INTeger, SiNGle precision, DouBLe precision, or STRing (INT, SNG, DBL, or STR).

Form:
DEFDBL letter list
DEFINT letter list
DEFSNG letter list
DEFSTR letter list

Example:
```
10 DEFDBL A-E
20 DEFDBL F-J
30 DEFSNG K-R
40 DEFSTR S-Z
```

DIM

Function: Allows specification of an array in one or two dimensions. It is required for tables in excess of 10 elements. A zero-relative element exists but must be specifically addressed. This statement declares the maximum subscript values and reserves the space.

Form: DIM variable name(dimensions) list

Example:
```
20 DIM A(25)
30 DIM B(10,10)
```

EDIT n

Function: Enter the EDIT mode by the command followed by a statement number. Move the cursor to the point of change. Insert by typing. Delete the current cursor location by pressing SHIFT/BKSP. Delete the previous character by pressing CTRL/BKSP. Return to BASIC program by pressing F8.

Form: EDIT line number range:
- null—the entire program
- line1–line2—between the lines, inclusive
- -line2 /' from the beginning to this line, inclusive
- line1- from this line to the end
- (.) this line

Example:
```
EDIT
EDIT 100-200
EDIT -200
EDIT 500-
EDIT
```

END

Function: Returns the program to the command level. The "Ok" prompt reappears.

Form: END

Example: 500 END

EOF

Function: Tests for End of File on RAM, cassette, or communications files.

Form: EOF(file number)

Example: 100 IF EOF(1) THEN CLOSE 1

ERL

Function: Gets line number where an error occurred. This instruction is used in conjunction with the ON ERROR GOTO instruction.

Form: ERL

Example:
```
100 ON ERROR GOTO 2000
:
2000 IF ERL = 20 THEN DV = 1:RESUME
```

ERR

Function: Gets the number of the error which occurred. This instruction is used in conjunction with the ON ERROR GOTO instruction.

Form: ERR

Example:
```
100 ON ERROR GOTO 2000
:
2000 IF ERR = 11 THEN DV = 1:RESUME
```

ERROR

Function: Simulates an error. Causes the MEW to function as if the requested error had been received. Important for testing purposes. Also important for built-in error codes which are not hardware dependent. These codes are normally drawn from the unassigned codes.

Form: ERROR numeric expression

Example: 200 ERROR 222

EXP

Function: Returns the exponential or natural antilog of the numeric expression. The numeric expression must be in the range of ±87.3365 or an overflow error will occur. EXP is the opposite of LOG.

Form: EXP(numeric expression)

Example: 100 PRINT EXP(22)

FILES

Function: Advises what files are stored within the RAM. The intrinsic functions (BASIC, TEXT, ADDRSS, and SCHEDL) will not be shown. This is done from the Command mode. This is also available from pressing Function Key 1.

Form: FILES

Example: FILES

FIX

Function: Truncates a value without rounding.

Form: FIX(numeric expression)

Example: 100 A = FIX(3276.33551)

FOR. . .NEXT

Function: Repeats program lines a specified number of times. The ellipsis can consist of a variable, expression, target expression, or incremental step. FOR. . . is used with NEXT.

Form:
```
FOR cv = iv TO fv STEP increment
:
NEXT cv
  cv = counter variable
  iv = initial value
  fv = final value
```

Example:
```
100 FOR N = 1 TO 10 STEP 2
:
150 NEXT N
```

FRE

Function: Determines the available memory.

Form: FRE(0)

Example:
```
100 PRINT FRE(0)
100 IF FRE(0) < 100 THEN END
```

GOSUB

Function: Transfers control to a subroutine, which must include a RETURN statement.

Form: GOSUB line number

Example:
```
100 GOSUB 1000
 .
 .
1000 FOR N = 1 TO 10
1010 PRINT N
1020 NEXT N
1030 RETURN
```

GOTO

Function: Unconditionally branches to a specified line.

Form: GOTO line number

Example: GOTO 1000

HIMEM

Function: Gets the High Memory Address

Form: HIMEM

Example: PRINT HIMEM

IF. . .THEN. . . ELSE

Function: Used to test logical conditions, allowing branches at logical differences.

Form: If relational or logical expression THEN command(s)1 ELSE command(s)2

Example:
```
100 IF A < 5 THEN 200 ELSE 300
200 IF (A = 10) OR (A = 20) THEN 500
300 IF (A = 15) AND (B = 10) THEN 700
```

INKEY$

Function: Used for a one-character strobe of the keyboard.

Form: INKEY$

Example: 100 Z$ = INKEY$:IF Z$ = "" THEN 100

INP

Function: Reads one byte of input from a port.

Form: INP(port number)

Example: A = INP(4)

INPUT

Function: Reads data (either alphabetic or numeric) from the keyboard.

Form: INPUT "prompt";variable list

Example:
```
100 INPUT A
200 INPUT "Enter Data: ";B
300 INPUT "Enter Name: ";NA$
```

INPUT

Function: Reads input from a RAM, cassette, communications, or modem file.

Form: INPUT # file number, variable list

Example: INPUT #1, R$

INPUT$

Function: Reads fixed number of characters from keyboard or file.

Form: INPUT$(numeric expression)
INPUT$(numeric expression, file number)

Example:
```
100 INPUT$(10)
200 INPUT$(10,1)
```

INSTR

Function: Locates a string within another string. The position of the first string within the second string is returned, making the instruction perfect for keyword location.

Form: INSTR (start position, search string, match string)

Example:
```
100 PRINT INSTR("ABCDEFGHIJ", "DEF")
200 PRINT INSTR(2, "ABCDEFGHIJ", "FGH")
300 PRINT INSTR(4, "ABCDEFGHIJ",L$)
400 PRINT INSTR(A$,B$)
```

INT

Function: Gets the whole number representation of a variable. The function will round and truncate as appropriate.

Form: INT(numeric expression)

Example:
```
100 PRINT INT(100.45)   (produces 100)
200 PRINT INT(99.65)    (produces 100)
200 PRINT INT(A)
```

IPL

Function: Defines program to be automatically executed during warm startup.

Form: IPL "filename"

Example: IPL "MEWPRG"

KEY

Function: Defines the function keys
Form: KEY function key number, string expression
Example: KEY 7,"?TIME$ + CHR$(13)

KEY LIST

Function: Lists the function key definitions in the format:
key 1 key 2
key 3 key 4
key 5 key 6
key 7 key 8
Form: KEY LIST
Example: KEY LIST

KEY ON/OFF/STOP

Function: Enables/disables the function key interrupts.
Form: KEY(function key number) ON/OFF/STOP
Example: KEY ON/OFF/STOP (operates for all keys)
KEY (n) ON/OFF/STOP
KEY ON—Activates the function key(s)
KEY OFF—Disables the function key(s)
KEY STOP—Disconnects the interrupt

KILL

Function: Deletes a RAM file.
Form: KILL "filename"
Example: KILL "MEWPRG"

LCOPY

Function: Copies the screen to a printer.
Form: LCOPY
Example: LCOPY

LEFT$

Function: Allows access to characters on the left side of a string.
Form: LEFT$(string expression,length)
Example: B$ = LEFT$(A$,5)

LEN

Function: Returns the length of a string variable.

Form: LEN(string expression)
Example: 100 A = LEN(A$)

LET

Function: Assigns the value of an expression to a variable. Not necessary to use.
Form: LET variable = expression
Example: 100 LET A = 5

LINE

Function: Draws the line from the coordinates x1,y1 to x2,y2. The X coordinates are numeric expressions in the range of 0 to 239 while the Y coordinates are numeric expressions in the range of 0 to 63. The instruction has a switch option for illuminating or extinguishing pixels, a code for developing a box, and a code for filling the box with color.
Form: LINE (x1,y1)-(x2,y2),switch,BF
Example:
```
100 LINE (40,40)-(60,40)
200 LINE (40,40)-(60,40),0
300 LINE (40,40)-(40,50),1
400 LINE (40,40)-(40,50),1,BF
```

LINE INPUT

Format: Inputs a string from the keyboard without regard for punctuation.
Form: LINE INPUT "prompt"; string variable
Example: 100 LINE INPUT A$

LIST

Function: Displays a program on the screen. Once begun, the program, if it is too long for the screen, will scroll. The scroll may be arrested by the PAUSE/BREAK key and restarted in the same manner. If only a part of a program is required, the inclusive lines can be specified with LIST (n–m). Also n or m can be separately specified (using the hyphen) to select only a portion of the program for display. Display of the entire program may be achieved by pressing the function key assigned to LIST (F5).
Form: LIST (line number or range)
Example: LIST Lists the entire program

LIST 100	Lists statement 100
LIST -100	Lists all until statement 100
LIST 100-200	Lists all between statements
LIST 100-	Lists from statement 100

LLIST

Same as LIST but targeted to the line printer; the same rules apply.

LOAD

Function: Used to load a program from a cassette, RAM, COM, or MDM. If device is RAM, include the extension .BA or .DO. The BASIC keyword is available as Function Key 2. The R operand will cause the program to be immediately run.

Form: LOAD "device:filename or configuration"

Example:
```
LOAD "RAM:PROG"
LOAD "CAS:PROG",R
LOAD "COM:78N1E"
LOAD "MDM:702E",R
```

LOADM

Function: Loads a machine language program from RAM or cassette.

Form: LOADM "RAM/CAS:filename"

Example:
```
LOADM "RAM:PROG"
LOADM "CAS:PROG"
```

LOG

Function: Returns the natural logarithm of a numeric expression which is greater than zero.

Form: LOG(numeric expression)

Example:
```
100 A = LOG(10)
```

LPOS

Function: Returns the current position of the printer head in the printer buffer.

Form: LPOS(dummy numeric expression)

Example:
```
100 PRINT "Head Position";LPOS(0)
```

LPRINT

Function: Prints the expression list on the printer. Expressions separated by semicolons will be printed contiguously.

Expressions separated by commas will be printed in zones, set at 14 characters. Numbers are printed with a trailing blank. Positive numbers are preceded by a blank. Negative numbers are preceded by a sign.

Form: LPRINT expression list

Example: LPRINT A$;B$,C$,D$

LPRINT USING

Function: Prints formatted data on the printer. A format string or print mask can be developed for punctuation, dollar signs, etc.

Form: LPRINT USING "format string";expression list

Example: 100 LPRINT USING "$$###,###.##";A
Examples are extensive—see the Model 100 Reference Manual for further detail.

MAXFILES

Function: Specifies the maximum number of files in use.

Form: MAXFILES = n

Example: 10 MAXFILES = 3
20 PRINT MAXFILES

MAXRAM

Function: Returns the amount of installed memory.

Form: MAXRAM

Example: CLEAR 500,MAXRAM

MDM ON/OFF/STOP

Function: Enables/disables the modem interrupts.

Form: MDM ON/OFF/STOP

Example:

100 MDM ON	Turns on the modem.
110 MDM OFF	Turns off the modem.
120 MDM STOP	Disconnects the interrupt.

MENU

Function: Returns control to the main menu.

Form: MENU

Example: 500 MENU

MERGE

Function: Combines two BASIC programs. Programs may be in

memory, in memory plus cassette, modem, or communications file. It is necessary to establish communications parameters for communications option. Refer to text and to Model 100 Reference Manual.

Form: MERGE "device:filename or configuration"

Example: MERGE "RAM:PROG2"

MID$

Function: Allows access to characters in the middle of a string. Replaces middle characters of a string.

Form: MID$(string expression, position, length)
MID$(string expression1, position, length) =
string expression2

Example: 100 PRINT MID$(A$,5,2)
200 MID$(A$,5,2) = "AB"

MOTOR

Function: Turns the cassette motor on and off.

Form: MOTOR ON/OFF

Example: 100 MOTOR ON
110 MOTOR OFF

NAME. . .AS

Function: Renames a file.

Form: NAME. . .AS

Example: NAME "MEWPRG.DO" AS "MEWPRG.BA"

NEW

Function: Wipes out the program in memory and clears all variables.

Form: NEW

Example: NEW

NEXT

See FOR. . .NEXT

ON COM GOSUB

Function: Defines a communications interrupt subroutine for incoming RS-232 communications. Assumes COM ON.

Form: ON COM GOSUB line number

Example: ON COM GOSUB 5000

ON ERROR GOTO

Function: Defines an error interrupt subroutine. Branch to the subroutine is taken if one of the hardware errors or one of the user errors is detected, as specified in the ERROR statement. Requires RESUME at the end of the subroutine.

Form: ON ERROR GOTO line number

Example: ON ERROR GOTO 4000

ON KEY GOSUB

Function: Defines function key interrupts. A separate subroutine is used for each function key, and must be specified in sequence. Assumes KEY ON command has been executed.

Form: ON KEY GOSUB line number list

Example: 100 ON KEY GOSUB 1000,2000,3000. . . .

ON MDM GOSUB

Function: Defines modem interrupt subroutine. Defines an interrupt for an incoming modem communication. Branch is taken to the MDM subroutine any time data are received. Requires that the MDM ON command has been executed.

Form: ON MDM GOSUB line number

Example: 100 ON MDM GOSUB 1000

ON TIME$ GOSUB

Function: Defines clock interrupt subroutine. When the appropriate designated time has been achieved, the subroutine will take control of the hardware.

Form: ON TIME$ = "time" GOSUB line number

Example: 100 ON TIME$ = "07:00:00" GOSUB 1000

ON. . .GOTO

Function: Unconditionally branches to a selected subroutine on the basis of the value of a variable.

Form: ON numeric expression GOTO line number list

Example: 100 ON A GOTO 1000,2000,3000

ON. . .GOSUB

Function: Performs a subroutine selected on the basis of the value of a variable.

Form: ON numeric expression GOSUB line number list

Example: 100 ON A GOSUB 1000,2000,3000

OPEN

Function: Opens a file for input, output, or appending. File may be in RAM, on CASsette, COMmunications, or MoDeM. Communications parameters must be specified, if used. Mode may be INPUT, OUTPUT, or APPEND.

Form: OPEN "device:filename or configuration"
FOR mode AS file number

Example: 100 OPEN "RAM:FILDAT.DO" FOR OUTPUT AS 1
110 OPEN "CAS:CASFIL" FOR INPUT AS 2

OUT

Function: Outputs a byte to a CPU port.

Form: OUT port number, byte value

Example: 10 OUT 55,100

PEEK

Function: Obtains a value directly from memory.

Form: PEEK(memory address)

Example: 100 A = PEEK(1000)

POKE

Function: Stores a value directly to memory.

Form: POKE memory address, byte value

Example: 100 POKE 67123,104

POWER

Function: Sets the automatic power down timing. The computer will automatically turn off after a period of numeric expression times 0.1 minutes. Default is 100, or 10 minutes.

Form: POWER numeric expression

Example: POWER 10

POWER CONT

Function: Prevent Automatic Power Down.

Form: POWER CONT
Example: 10 POWER CONT

POWER OFF

Function: Immediate Power Down.
Form: POWER OFF,RESUME
Example: 100 POWER OFF,RESUME

PRESET

Function: Turns off an LCD pixel. X and Y coordinates in the accepted range are required.
Form: PRESET (X-coordinate, Y-coordinate)
Example: PRESET (32,120)

PRINT

Function: Prints data on the LCD screen. Screen location may be designated within the acceptable range of 0–320.
Form: PRINT expression list
Example:
```
100 PRINT
110 PRINT A;A$
120 PRINT A,A$
130 PRINT @249, "A"
```

PRINT

Function: Writes sequential output to a file.
Form: PRINT # file number, expression list
Example:
```
100 OPEN "RAM:" FOR OUTPUT AS 1
100 PRINT #1,A$
```

PRINT USING

Function: Prints formatted data on the screen. A format string or print mask can be developed for punctuation, dollar signs, etc.
Form: PRINT USING "format string";expression list
Example: 100 PRINT USING "$$###,###.##";A
Examples are extensive—see the Model 100 Reference Manual for further detail.

PRINT # USING

Function: Writes formatted information to file. The rules and constraints are the same as for the PRINT USING, except the output is a file.

Form: PRINT # USING "format string"; expression list
Example: PRINT #1 USING "$$###,###.##";A

PSET

Function: Turns off an LCD pixel. X and Y coordinates in the accepted range are required.
Form: PSET (x-coordinate, y-coordinate)
Example: PSET (32,120)

READ

Function: Accesses internally stored DATA lines and retrieves the data as designated variables—customarily array elements.
Form: READ variable list
Example:
```
100 DATA 0,4,2,6,"Ken"
110 READ A,B,C,D,N$
```

REM

Function: Includes REMarks within a program. This function can also be generated by an apostrophe.
Form: REM
Example:
```
10 REM This is a remark
20 ' So is this
```

RESTORE

Function: Resets the pointer to the first DATA line contained in the program and allows DATA lines to be reread. Alternately, where multiple sets of DATA statements are involved, a specific line number at the head of a list may be specified.
Form: RESTORE line number
Example:
```
100 RESTORE
200 RESTORE 500
```

RESUME

Function: Resumes execution after error or interrupt.
Form: RESUME
Example: 2010 RESUME

RETURN

Function: Used to return from a subroutine. It enables a return

to a particular location given the specification of a line number.

Form: RETURN

Example: 2010 RETURN

RIGHT$

Function: Allows access to characters on the right side of a string.

Form: RIGHT$(string expression,count)

Example: PRINT RIGHT$(A$,3)

RND

Function: Returns pseudo random number in the range of 0–1. Numbers are generated from a fixed table and must be interpreted for use. If the numeric expression is 0, then RND returns the last random number generated.

Form: RND(numeric expression)

Example:
```
100 PRINT RND(1)
110 A = RND(0)
```

RUN

Function: Causes the program in memory to be executed. If the program is loaded from cassette tape and the RUN option follows the filename, the computer will read the program from the tape and begin execution. Available as a function key (F4). Operand R keeps the files open upon program completion.

Form:
```
RUN "device:filename or configuration",R
RUN
RUN line number
```

Example:
```
RUN "RAM:MEWPRG.BA",R
RUN
RUN 100
```

RUNM

Function: Loads and executes a machine language program.

Form: RUNM "CAS or RAM:filename"

Example: RUN "RAM:MEWPRG"

SAVE

Function: Used to record a copy of the program located in mem-

ory onto a cassette tape. The BASIC keyword is available as Function Key 3. The A operand causes the file to be saved as an ASCII file.

Form: SAVE "device:filename or configuration",A

Example: SAVE "RAM:MEWPRG",A

SAVEM

Function: Stores a machine language program.

Form: SAVEM "CAS or RAM:filename", start address, end address, entry address

Example: SAVEM "CAS:MEWPRG",50000,50305,50020

SCREEN

Function: Locks/unlocks the LABEL line.

Form: SCREEN on/off

Example:
```
100 SCREEN 0,0 (on)
110 SCREEN 0,1 (off)
```

SGN

Function: Returns the algebraic sign of a numeric expression. A −1 is returned for negative numbers, 0 for 0 numbers, and 1 for positive numbers.

Form: SGN(numeric expression)

Example: 100 A = SGN(B)

SIN

Function: Returns the trigonometric sine of a numeric expression.

Form: SIN(numeric expression)

Example: 100 S = SIN(1.5)

SOUND

Function: Used to generate a musical note within five octaves, establishing the frequency and specific duration of that note.

Form: SOUND pitch,length

Example: 100 SOUND 6642,200

SOUND ON/OFF

Function: Enables or disables sound.

Form: SOUND ON or OFF

Example: 100 SOUND OFF
200 SOUND ON

SPACE$

Function: An expandable constant that allows specification of a string of blank characters at execution time.

Form: SPACE$(length)

Example: PRINT SPACE$(50);"Message"

SQR

Function: Returns the square root of a numeric expression.

Form: SQR(numeric expression)

Example: 100 A^2 = B^2 - 4AC
110 D = SQR(A)

STOP

Function: Stops the program execution, prints the break message, and returns a program to command level. The program can be restarted with a CONTinue command.

Form: STOP

Example: 500 STOP

STR$

Function: Used to convert a numeric value to a string variable.

Form: STR$(numeric expression)

Example: 100 PRINT "$" + STR$(AMT) + ".00"

STRING$

Function: Defines a string of characters whose maximum length is 255 characters.

Form: STRING$(length,character)

Example: 100 PRINT STRING$(70, "-")

TAB(n)

Function: Used for horizontal spacing on display or printer.

Form: TAB(numeric expression)

Example: 100 PRINT TAB(15);"Message"

TAN

Function: Returns the trigonometric tangent of a numeric expression.

Form: TAN(numeric expression)
Example: 100 A = TAN(B)

TIME$

Function: Gets and keeps track of the current time.
Form: TIME$
Example:
```
100 TIME$ = "07:00:00"
110 PRINT TIME$
```

TIME$ ON/OFF/STOP

Function: Enables/disables time interrupt. Enable turns on the interrupt. Disable turns it off. Stop disconnects it.
Form: TIME ON/OFF/STOP
Example: 100 TIME OFF

VAL(x$)

Function: Used to determine the value of a number that has been developed in alphabetic form.
Form: VAL(string expression)
Example: 100 A = VAL(A$)

VARPTR

Function: Gets the address of a variable.
Form: VARPTR(variable name)
Example: 100 A = VARPTR(A$)

APPENDIX B

Using Non-Radio Shack Printers

As may be common knowledge at this point, Radio Shack printers do not require the Linefeed (LF) character, ASCII 10. Therefore, with the printer cable (Part 26-1409, $15), it is possible to list programs and documents (.DO) on Radio Shack printers—only. The minute you try it with a couple of other popular printers (I tried the Epson MX-80 and the NEC 3350), you suddenly find that your entire document prints on a single line, not a good situation at all. You can get around it by placing a "GRPH-slash" combination at the end of each line, but that's work.

The following is a BASIC program designed to print M100 documents. In constructing the program, a few things were learned about how the M100 stores documents. The 40 contiguous screen characters, if not fully occupied, are not matched in memory. If a long word (such as an eight-letter word) could not be accommodated in the available space (e.g., five characters), it is placed on the second line of the LCD (Liquid Crystal Diode) screen by the ROM screen management routines. However, in memory (a RAM file), the data is contiguous, separated only by the space which would separate words.

The program scans the line that is poised for printing to ensure that trailing words, which are longer than the available space, are moved to the next line. It also prevents a leading space in a subsequent line should the final character of a word fall at the right margin. If you specify a width of 40, the printout should be just as it appears on the LCD screen. The program offers options for hardcopy and display viewing, or display viewing alone. You may repeat it as necessary, or until the string space allocation (CLEARed of 5000 bytes) has been used.

With this routine you will be able to directly print the documents you have prepared on the train, plane, or bus (when you weren't answering other passengers' questions about what that thing is you have in your lap).

```
10 CLS
20 CLEAR 5000
30 CLS
40 INPUT "HARDCOPY (Y/N)?";Q$
50 IF (Q$="Y") OR (Q$="y") THEN Q$="Y":GOTO 70
60 Q$ = "N"
70 FILES
80 PRINT "Enter file name to"
90 INPUT "be printed: ";FILE$
100 WHICH$ = "RAM:" + FILE$ + ".DO"
110 OPEN WHICH$ FOR INPUT AS 1
120 FOR N = 1 TO 5000
130     LINEINPUT #1,TXT$
140     IF LEN(TXT$)<41 THEN 150 ELSE 170
150     IF Q$="Y" THEN LPRINT TXT$;CHR$(10)
160     PRINT TXT$:GOTO 330
170     NR=LEN(TXT$):X=1
180     IF NR>40 THEN L=40 ELSE L=NR
190     IF MID$(TXT,X+L,1)=" " THEN 230
200     L=L-1
210     IF (L<1) AND (NR>0) THEN L=NR:GOTO 230
220     GOTO 190
230     PARED$=MID$(TXT$,X,L)
240     FOR Z = 1 TO L
250         IF MID$(PARED$,Z,1)<>" " THEN 270
260     NEXT Z
270     PARED$=MID$(PARED$,Z,L-Z+1)
280     PRINT PARED$
290     IF Q$="Y" THEN LPRINT PARED$;CHR$(10)
300     PARED$=""
310     NR=NR-L
320     IF NR>0 THEN X=X+L:GOTO 180
330     IF EOF(1) THEN 350
340 NEXT N
350 CLOSE 1
360 IF Q$= "Y" THEN LPRINT CHR$(10);CHR$(12);CHR$(10)
370 PRINT
380 F=FRE("")
390 INPUT "ANOTHER (Y/N)";Q$
400 IF (Q$="Y") OR (Q$="y") THEN 30
410 MENU
```

APPENDIX C

Computer-to-Computer Communications

The TRS-80 Model 100 is a most amazing computing machine. However, like most new entries into the marketplace, it comes without a few directions. And, you will suddenly discover, there are a few things you'd like to do which you simply cannot do as things are constituted. One of these is the process of transferring data to and from another computer. Of course Radio Shack would like for you to connect the Model 100 to another Radio Shack computer, and the directions are marginally sufficient about how to do that. But, for working with another machine, you're pretty much on your own. Faced with the need to so connect my Model 100 to my IBM PC, I was able to do so only after some extreme exasperation and struggling to figure it out myself, for nobody had any answers.

In order to communicate between any two computers it is necessary to obtain two things: an RS-232C cable (Part 26-1490, $30) and a Null Modem (Part 26-1496, $30). The *null modem* is simply a conditioning device. Its purpose is to reverse pins 2 and 3 to facilitate computer-to-computer transmission. This permits pin 2 (Transmit Data) of each computer to communicate with pin 3 (Receive Data) of each computer.

The cable, however, is another story. The Radio Shack cable has a male connection on both ends and the plug on those male connections is molded in such a way as to facilitate connection at the rear of the Model 100. Other available cables are molded differently and will not fit. Unfortunately, the RS-232C connector at the back of non-Radio Shack computers is also male, meaning that it expects a female connection. Unless you are prepared to have a male-to-female cable especially pre-

pared, this means another $30 expenditure for part 26-1495, RS-232C Cable Connector (commonly called a *sex-change connector*), which is double female. These are the cable specifications:

Cable Fabrication Specifications

PIN NUMBERS	SIGNAL
1 to 1	Protective Ground
2 to 3	Transmit Data
3 to 2	Receive Data
4 to 4	Request to Send
5 to 5	Clear to Send
6 to 6	Data Set Ready
7 to 7	Signal Ground
8 to 8	Carrier Detect
15 to 15	I/O Transmit SET
17 to 17	Receiver Clock
20 to 20	Data Terminal Ready
24 to 24	Transmit Clock

The only disadvantage is that when the assembly is plugged into the back of your desktop computer, it is vulnerable to any motion in the vicinity. It cannot be plugged directly into the Model 100, because the plug is molded differently and will not fit in the available space.

Next comes the communications tasks. The following details the step-by-step procedure for transmitting data (uploading) from the Model 100 to the IBM PC. The communications protocols for both machines bear explanation:

The Radio Shack Protocol is: 57N1D,10 pps, where:

5 = 1200 baud

7 = number of bits in an ASCII character

N = no parity check

1 = number of stop bits

D = XON/XOFF is disabled

10 pps = 10 pulses per second (sufficient speed)

The IBM PC protocol, executed through a MODE instruction is:

Mode 12,N,7,1—where:

12 = 1200 baud
N = no parity
7 = number of bits in the ASCII character
1 = number of stop bits

In both procedures, it is imperative to follow the steps precisely, specifically as regards the pressing of the ENTER (<E>) key.

Transfer Data From the Radio Shack Model 100 to the IBM PC (or PC/XT)

M100	PC
1. Move cursor to TELCOM <E>.	
2. Press F3 (Stat) Enter 57N1D,10 pps <E>.	
3. Enter MODE COM1:12,N,7,1 <E>.	
4. Press F4 (Term)	
5. Press F3 (Up) <E>. Computer: File to Upload? Enter name <E>. Computer: Width: Enter 80 (NO <E>).	
	6. Enter COPY AUX SWAP.TXT <E>. (or whatever document name)
7. <E>. The copy takes place. File(s) copied message on PC.	
8. Press F8 (Bye).	
9. Computer: DISCONNECT? Respond Y.	
10. Press F8 (Menu).	

Transfer Data From the IBM PC (or PC/XT) to the Radio Shack Model 100

PC	M100
	1. Move cursor to TELCOM <E>.
	2. Press F3 (Stat) Enter 57N1D,10 pps <E>.
3. Enter MODE COM1:12,N,7,1 <E>.	
	4. Press F4 (Term).
	5. Press F2 (Down). Computer: File to Download? Enter name (NO <E>).
6. Enter COPY SWAP.TXT AUX (or whatever document name) (NO <E>).	
	7. <E>.
8. <E>. The copy takes place, and shows on M100 screen. File(s) copied message on PC.	
9. Press F8 (Bye).	
10. Computer: DISCONNECT? Respond Y.	
11. Press F8 (Menu).	

Index

Abstracts, 62, 68
AC adapter, 15
Access code, 80
Access, dial-up, 81
Acoustic connection, 71
Acoustic coupler, 7, 36, 42, 43, 70, 165
Adapter
  AC, 15
  null modem, 43
  RS-232 communication, 48
  sex change, 43
ADDRESS, 17
Address block, 83
Address organizer, 39
Address, machine, 183
ADDRSS, 9, 35, 39
Adjustment, vernier, 7
ADRS.DO, 39, 40, 41, 52, 53, 70, 77
Algebraic sign, 193
Algorithms, 94
Alphabetic data, 88
Alphabetic variables, 88, 129
ANS/ORIG, 52, 70, 71
APPEND, 166
Appending, 134
Arithmetic/logic unit (ALU), 88
Array, 138
  multidimensional, 138, 139
  unidimensional, 138
Ascending sequence, 150
ASCII, 5, 96
  character set, 48
  codes, 31, 98
  token, 116
  values, 98
Asynchronous communication, 49
Asynchronous mode, 82
ATN, 219, 221
Audio tones, 17
Auto Log-On Commands, 53
Auto-Dialer, 7
Automatic dialing, 42, 52, 55
AUX connector, 29
AVERAGE, 198
Aviation, 68

Backup copies, 108
Bar code reader, 8
BASIC, 5, 12, 13, 35, 87
  Microsoft, 10
  prompt, 13, 91, 92, 107
Batteries
  alkaline, 10
  nickel cadmium, 10
Baud, 8, 52, 83
Baud rate, 44, 48
BEEP, 98, 188, 221
Bell System, 43
Bibliographic information, 62, 65
Binary digit, 8, 48
Binary search, 144, 150
Bit, stop, 44, 49
BKSP key, 107
Blinking cursor, 13
Blinking display, 117
Block graphics, 216
Block
  Address, 83
  End, 83
  Message, 83
  Text, 83
  Text Control, 81, 83
Boldface, 31
Branch
  conditional, 104
  unconditional, 99
BREAK, 6, 19, 22, 39, 41, 109
Broadcasting, 63, 67
BS ERROR, 142
Buffer, 21, 165
Buffer, PASTE, 22, 27
BULLET, 76
Bulletin boards, 64
Bye, 48, 52
Bytes, 88
Bytes free, 16, 91

Cable
connections, 29
male-to-female, 245
null modem, 42
outboard direct connection, 36
Calculations, mathematical, 8
Calculator mode, 94, 99
CALL, 179, 183, 188, 221
CAPS LOCK key, 5
Carriage return, 115
Carriage-return, linefeed, 97
Cassette drive, 87
CASsette files, 155
Cassette interface, 8
Cassette tape, 17, 29, 96, 108
CB, 76
CDBL, 182, 186, 221
Central power switch, 10
Centronics parallel port, 31
Centronics parallel printer, 7
Characters, graphic, 33, 118
Charge information, 46
Charts, music, 63
Chemical substances, 63
Chip, 8, 88
Chips, processor, 49
CHR$, 118, 126, 222
CINT, 183, 186, 222
CLEAR, 133, 186, 222
Clear screen, 98
CLOAD, 96, 97, 188, 222
CLOAD?, 188, 222
CLOADM, 188, 222
Clock, military, 14
CLOSE, 15, 165, 188, 222
Closed subroutine, 105
CLS, 98, 118, 126, 223
CMOS, 8
CODE key(s), 5, 204
access, 80
ASCII, 31, 98
graphic, 118
COM files, 155, 168
COM OFF, 186, 223
COM ON, 186
COM STOP, 186, 223
COM/MDM OFF, 177
COM/MDM ON, 177
COM/MDM STOP, 177
Command key, 13
functions, 38, 41
Commands, 124
Auto Log-On, 53
Commands *(Contd.)*
machine language, 17
Commodity futures, 67
Common Text Message, 81
Communications
asynchronous, 49
capabilities, 36
electronic, 75
interruptions, 177
lines, 112
media, 51
onboard process, 36
package, 35
parameters, 48
protocol, 44, 47, 48, 51, 70
RS-232 interface, 43
serial, 49
service, 79
Compound conditionals, 159
Compound interest, 201
Compressed mode, 96, 119
CompuMoney, 53
CompuServe, 9, 53, 59, 60, 61, 69, 70, 72, 73, 74, 76
CON ON, 223
Concatenation, 134
variable, 131
Conditional branching, 104
Conditionals
compound, 159
error, 102
Connection
acoustic, 71
cable, 29
direct, 42, 52, 70
RS-232, 165
Connector
AUX, 29
EAR, 29
pin, 8
REM, 29
RS-232, 31, 47, 51, 70
sex-change, 246
Console mode, 94
Consolidated Gizmo, 207
Constant data, 130
CONT, 101, 223
Control
tone, 30
volume, 30
Controls
external, 87
internal, 87
Copy, 18, 22

COPY, 38, 41
COS, 190, 196, 219, 223
COT, 81, 83
Coupler, acoustic, 7, 36, 42, 43, 70, 165
CRSLIN, 219, 224
CSAVE, 96, 97, 188, 223
CSAVEM, 188
CSNG, 182, 186, 224
CSRLIN, 206
CSVEM, 223
CTRL, 19
CTRL key, 4, 107
Cursor keys, 6, 12, 19
Cursor, blinking, 13
Cut, 22
CUT, 18, 27, 38, 41
Cystic Fibrosis Fund, 90, 99

Daisy-wheel printer, 33
DATA, 136, 188, 224
  lines, 135, 155
  statement, 134
Data
  alphabetic, 88
  constant, 130
  dropout, 30
  numeric, 88
  receive, 245
  transient, 130
  transmit, 245
Database, 61, 79
  economic, 63
  financial, 63
  financial business, 67
DATE$, 14, 189, 224
DAY$, 14, 189, 224
DEFtype, 187, 225
DEL key, 27
DEL/BKSP key, 6
DELETE, 110
Deletion, 27
Derived functions, 197
Dialing
  automatic, 42, 55
  automatic procedure, 52
  semi-automatic, 42
Dial Pulse rate, 44, 49
Dial-up access, 81
Digit, binary, 8, 48
DIM, 139, 187, 225
Diminished print, 31
DIN, 43
DIR, 43
DIR/ACP, 43, 48
Direct connect modem, 7, 177
Direct connection, 42, 52, 70
  outboard, 36
Direct dial network, 54
Direct mode, 94, 99
Directory, 12
Disconnect, 57
Diskette drives, 17
Display
  blinking, 117
  liquid crystal, 7, 88
Display screen, 112
DJNS, 77, 78
Double size mode, 119
Double strike mode, 119
Double-precision variable, 130
Dow Jones, 9, 53, 59, 61, 69, 76, 77, 79
Down, 47, 51, 56
Downloading, 36, 47, 51, 56, 79
Drive
  cassette, 87
  diskette, 17
Duplex
  full, 9, 47, 52
  full service, 47
  half, 9, 47, 52
  half service, 47
Duplication, 27

E notation, 102
E-COM, 80, 81, 82
EAR connector, 29
Echo, 46, 48, 51
Economic database, 63
EDIT, 99, 107, 124, 225
Editor, 107
Editor, TEXT, 96
Electric Pencil, 9
Electronic mail, 42, 60, 74
Electronic newsletters, 67
Electronic telephone systems, 49
Elements, picture, 7
ELSE, 143
EMAIL, 76
Emissions, spurious, 79
Emphasized mode, 119
END, 101, 125, 150, 226
End Block, 83
Energy, 64
Enlarged print, 31

ENTER key, 6, 12, 15, 44
Entertainment, 74
Entry mode, 42, 44
Entry, manual, 50
EOF, 176, 188, 226
Epson MX-80, 7, 18, 97, 118
Equality, 104
ERL, 177, 188, 226
ERROR (ERR), 177, 188, 226
Error conditions, 102
Error trapping, 122
Error, syntactical, 129
Errors, 174
ESC key, 4
Exchange sort, 151, 154
EXP, 197, 219, 226
Exponential notation, 102
External controls, 87
External modem, 47

FEEDBK, 76
FILE, 125
Filename, 166
FILES, 227
Files, 12, 13, 92
  CASsette, 155
  COM, 155, 168
  MDM, 155, 165
  modem, 168
  RAM, 154, 155, 161, 165, 168
Financial database, 63, 67
Find, 38, 40, 41, 42, 44
Find function, 17
FIX, 187, 227
Flag, 152
FOR, 125
For. . .NEXT, 106, 108, 111, 227
FRE, 186, 227
FRE(0), 102
Full duplex, 9, 47, 52
Full duplex mode, 83
Functions
  Command key, 38, 41
  derived, 197
  find, 17
  INSTR, 143
  intrinsic, 8, 9
  POWER, 182
  renumbering, 108
  string, 126

Garbles, 79
Glitches, 79
Globe and Mail, 65
GOSUB, 106, 122, 125, 227
GOTO, 99, 100, 101, 104, 121, 125, 143, 228
GRaPH key, 33
Graphics, 7
  block, 216
  characters, 33, 118
  codes, 7
  printer, 33
  in 3-D, 209
Greater than, 104
Greater than or equal to, 104
Group label, 83
GRPH keys, 5, 204

Half duplex, 9, 47, 52
Half duplex service, 47
Handling, table, 154
Hardcopy
  output, 155
  printer, 115
  printout, 112
Header, message, 83
Hierarchy, 73
HIMEM, 102, 186, 228
Home information, 74
Horizontal span, 138
Horses, thoroughbred, 63

IBM PC, 20
IF, 126
IF. . .THEN. . .ELSE, 228
Immediate mode, 99
Indented margins, 11
Indirect mode, 94
Inequality, 104
Information
  bibliographic, 62, 65
  charge, 46
  commodities, 64
  financial, 74
  home, 74
  medical, 65, 66
  nonnumeric, 14
  on-line service, 59
INKEY$, 123, 124, 189, 228
INP, 184, 188, 228
Input, 87

INPUT, 101, 121, 126, 166, 228
INPUT #, 168, 188, 229
INPUT$, 188, 189, 229
Input/Output statements, 126
INST, 188
INSTR, 229
INSTR function, 143
Instruction separator, 105
INT, 147, 187, 229
Integer variable, 130
Integration, 12
Interest
  compound, 201
  simple, 201
Interface
  cassette, 8
  parallel printer, 7
  RS-232 communication, 43
  serial, 31
Intermediate mode, 14
Internal controls, 87
Interpretive compiler, 10
Interruptions, communication, 177
Interrupts, 172
Intrinsic functions, 8, 9
Intrinsic programs, 12, 92
Intrinsic software, 6
Intrinsic software systems, 35
INTRO, 78
Inventory table, 157
IPL, 218, 219, 229
Iteration, 108

Jack, modular phone, 70
Justification, line, 11

KEY LIST, 178, 186, 230
KEY OFF, 178, 187, 230
KEY ON, 178, 187, 230
KEY STOP, 178, 187, 230
Keys
  BKSP, 107
  CAPS LOCK, 5
  CODE, 5, 204
  command, 13
  command functions, 38, 41
  CTRL, 4, 107
  cursor, 19
  cursor control, 6
  cursor movement, 12
Keys *(Contd.)*
  DEL/BKSP, 6
  ENTER, 6, 12, 15, 44
  ESC, 4
  GRaPH, 33
  GRPH, 5, 204
  left arrow, 12
  PASTE, 6
  PRINT, 6, 31
  right arrow, 12
  SHIFT, 5, 6, 18, 31, 107
  TAB, 4, 111
Keyboard, 4
  numeric, 5
  QWERTY, 5
  strobe, 124
KILL, 16, 169, 186, 230

LABEL, 13, 17, 18, 39, 41
LCOPY, 113, 186, 230
Left arrow key, 12
LEFT$, 142, 189, 230
Legal cases, 62, 69
Legal research service, 66
LEN, 189, 230
Length, 109
Length, word, 44, 48, 49
Less than, 104
Less than or equal to, 104
LET, 14, 100, 126, 231
Lfnd, 38, 41, 42
LINE, 205, 219, 231
Lines
  communications, 112
  DATA, 135, 155
  screen, 7
  sentinel, 137
  telephone, 48, 52
LINE INPUT, 188, 189, 231
Line justification, 11
Line printer, 31
Line terminator plug, 69
Linefeed, 115
Linefeed carriage-return, 97
Liquid Crystal Display (LCD), 7, 88
  screen, 98
  wristwatch, 7
List, 13, 92
  sequential, 138
LIST, 96, 97, 108, 125, 231
LLIST, 125, 232

Load, 13, 17, 92
LOAD, 96, 125, 232
LOADM, 188, 232
LOG, 197, 219, 232
Log on, 46
  procedures, 70
  sequence, 53
Logic, 88
Looping, 104
LPOS, 206, 219, 232
LPRINT, 112, 126, 136, 232
LPRINT USING, 126, 188, 233
LPRINT # USING, 188
LPT, 31

Machine address, 183
Machine language commands, 17
Magnetic media, 155
Mail, electronic, 42, 60, 74
Mailgram, 80
Main Processing Subsystem, 80
Maintenance, table, 154
Male-to-female cable, 245
Management Operations Center, 81
Manual entry, 50
Margins, indented, 11
Marketing, 67
MasterCard, 73
Mathematical calculations, 8
Mathematics, 193
Math operators, 102
Matrix, pixel, 203
Matrix printer, 88
MAXFILES, 165, 187, 233
MAXRAM, 102, 187, 233
MDM files, 155, 165
MDM OFF, 187, 233
MDM ON, 187, 233
MDM STOP, 187, 233
MEAN, 198
Media
  communications, 51
  magnetic, 155
  tape cassette, 29
Medical information, 65, 66
Memory, 49, 88
  parity, 49
  size, 102
  random access, 8
  read only, 8, 35
  volatile, 10
MENU, 12, 13, 150, 187, 233
Menu, 18, 38, 40, 41, 44, 108
  TELCOM, 56
  top, 73
MERGE, 186, 187, 233
Message blocks, 83
Message header, 83
Microcompressor, 8
Microsoft BASIC, 10
MID$, 142, 189, 234
Military clock, 14
Modes, 166
  asynchronous, 82
  calculator, 94, 99
  compressed, 96, 119
  console, 94
  direct, 94, 99
  double size, 119
  double strike, 119
  emphasized, 119
  entry, 42, 44
  full duplex, 83
  immediate, 99
  indirect, 94
  intermediate, 14
  OUTPUT, 168
  synchronous, 82
  terminal, 42, 45, 46, 48, 49, 50, 52, 54, 55, 56, 57, 71
  TEXT, 11
MODEM, 43
Modems, 47, 48, 49, 52, 70, 83
  direct connect, 7, 177
  external, 47
  files, 168
  null, 43, 245
  null adapter, 43
  onboard, 36
Modular phone jack, 70
Modules, 155
Monitor, 112
More, 42
MOTOR, 186, 188, 234
MOVING AVERAGE, 198
MPS, 80
Multidimensional array, 138, 139
Municipal debt, 68
Music charts, 63

NAME. . .AS, 186, 234
Negative numbers, 183

Nested subroutine, 123
Nesting, 110
Network, direct dial, 54
NEW, 125, 234
Newsletters, electronic, 67
NEXT, 126, 234
Nonnumeric information, 14
Notation
  E, 102
  exponential, 102
  shorthand, 116
  standard scientific, 102
NOTE.DO, 36, 38
Null modem, 43, 245
  adapter, 43
  cable, 42
Null string, 124
NUM, 5
Numeric keyboard, 5
Numeric variables, 88, 89, 129

ON COM, 177, 178
ON COM GOSUB, 187, 234
ON ERROR, 178
ON ERROR GOTO, 187, 235
ON. . .GOSUB, 121, 126, 236
ON. . .GOTO, 121, 126, 235
ON KEY, 178
ON KEY GOSUB, 187, 235
ON MDM, 177, 178
ON MDM GOSUB, 187, 235
ON TIME$, 173
ON TIME$ GOSUB, 187, 235
On/off switch, 10
Onboard communications process, 36
Onboard modem, 36
OPEN, 165, 188, 236
Operators
  math, 102
  relational, 103
OPR, 74
Organizer
  address, 39
  schedule, 36
ORIG, 46, 50, 70, 71
OUT, 184, 188
Outboard direct connection, 36
OUTPUT, 166
Output, 88
  hardcopy, 155
  printed, 7
OUTPUT mode, 168
OUY, 236

Parallel printer interface, 7
Parameters, communications, 48
Parity, 44, 49, 70
  memory, 49
Parkinson's Law, 133
Password, 46, 53, 55
PASTE, 18, 38, 41
  buffer, 22, 27
  keys, 6
PAUSE, 6, 19, 97, 105, 118
PEEK, 183, 188, 236
Personnel schedule, 35
Picture elements, 7
Pin connectors, 8
Pitch, 109
Pixels, 7, 203
PLAY, 29, 96
PLAY button, 30
Plug, line terminator, 69
POKE, 183, 189, 236
Port, centronics parallel, 31
POS, 206, 219
Positions, screen, 6
POWER, 187, 236
POWER CONT, 182, 187, 236
POWER function, 182
POWER OFF, 182, 187, 237
PRESET, 204, 219, 237
Prev, 47, 51
PRINT, 18, 39, 41, 100, 112, 118, 126, 136, 237
PRINT #, 168, 189, 237
PRINT keys, 6, 31
PRINT USING, 120, 126, 184, 237
PRINT # USING, 168, 189, 237
Print
  diminished, 31
  enlarged, 31
  mask, 120
  size, 119
Printed output, 7
Printer, centronics parallel, 7
Printers
  daisy-wheel, 33
  graphics, 33
  hardcopy, 115
  line, 31
  matrix, 88
  parallel interface, 7

Printers *(Contd.)*
serial, 47, 51, 165
Radio Shack, 30
Processor chips, 49
Processor
onboard communications, 36
text, 11, 29
TEXT, 31, 38, 87
word, 11
Programming, BASIC language, 35
Prompt, BASIC, 13, 91, 92, 107
Protocol
communications, 44, 47, 48, 51, 70
transmission, 83
PSET, 204, 219, 238

QUERY, 78
Quit, 42
QWERTY, 4, 5

Radian, 194
RAM files, 154, 155, 161, 165, 168
Random Access Memory (RAM), 8, 10, 134, 181
RANGE, 198
Rate
baud, 44, 48
dial Pulse, 44, 49
READ, 135, 136, 189, 238
Read Only Memory (ROM), 8, 35, 87
Receive data, 245
RECORD, 96
Relational operators, 102
REM, 100, 126, 238
REM connector, 29
Renumbering function, 108
Reset button, 79
RESTORE, 136, 155, 189, 238
RESUME, 182, 187, 238
RETURN, 105, 106, 121, 126, 238
Reverse scanner, 51
Reverse video, 12, 91
Reversed string variables, 14
Right arrow key, 12
RIGHT$, 142, 189, 239
RND, 150, 187, 239
Rotary telephone systems, 49
RS-232, 7
communications adapter, 48
communications interface, 43
connector, 31, 47, 51, 70, 165
Run, 13, 92
RUN, 96, 97, 98, 99, 108, 125, 186, 239
RUNM, 186, 239

SAM, 81, 83
Save, 13, 17, 92
SAVE, 125, 239
Save to, 30
SAVEM, 186, 240
SCHEDL, 9, 17, 35, 36, 38
SCHEDL.DO, 38
Schedule organizer, 36
Scientific instruction set, 193
SCREEN, 180, 186, 240
Screen, 7
clear, 98
display, 112
LCD, 98
line, 7
positions, 6
Scripsit, 9
Scroll, 180
Scrolling, 6, 15
Search, binary, 144, 150
Securities, corporate, 68
Security, 46
Semi-automatic dialing, 42
Semiconductor, 8
Sentinel, 161
Sentinel line, 137
Sequence
ascending, 150
log-on, 53
Serial communications, 49
Serial interface, 31
Serial printer, 47, 51, 165
Services
communications, 79
CompuServe, 73
full duplex, 47
half duplex, 47
legal research, 66
on-line information, 59
Set
ASCII character, 48
scientific instruction, 193
Sex change adapter, 43
Sex-change connector, 246
SGN, 193, 219, 240
Shell-Metzner sort, 151
SHIFT, 19, 41
SHIFT key, 5, 6, 18, 31, 107
Shorthand notation, 116

Simple interest, 201
SIN, 190, 219, 240
Single Address Message, 81
Single-precision variable, 130
Software, intrinsic, 6, 35
Sort
  exchange, 151, 154
  Shell-Metzner, 151
SOUND, 109, 126, 155, 156, 189, 240
SOUND OFF, 187, 240
SOUND ON, 187, 240
Source, The, 9
SPACE$, 144, 189, 217, 240
Span, horizontal, 138
Speaker, 79
Specialized reports, 62
Speed transmission, 8, 47
Speed typing, 4
SPO, 80
Spurious emissions, 79
SQR, 194, 219, 241
Standard and Poor, 64
Standard scientific notation, 102
Starter kit, 60
Stat, 44, 49
Statements, 125
  DATA, 134
  input/output, 126
Statistics
  economic, 64
  vital, 4
Status, 44
STEP, 109
STIP/CONT, 126
STOP, 101, 241
Stop bit, 44, 49
STR$, 182, 189, 241
Strings, 142
  functions, 126
  null, 124
  variables, 14, 80, 130
  variables reversed, 14
STRING$, 144, 189, 216, 241
Strobe, keyboard, 124
Subroutines
  closed, 105
  nested, 123
Subsystem, Main Processing, 80
Suffix, 16
Switch
  central power, 10
  on/off, 10
Switching arrangement, 117
Sequential list, 138
Synchronous mode, 82
Syntactical error, 129
Systems
  electronic telephone, 49
  intrinsic software, 35
  rotary telephone, 49

TAB, 127, 241
TAB key, 4, 111
Tables
  handling, 154
  inventory, 157
  maintenance, 154
Tabulation, 116
TAN, 193, 196, 219, 241
TANgent, 196
TELCOM, 9, 35, 36, 40, 42, 44, 46, 48, 50, 52, 70, 71
  equipment, 43
  menu, 56
Telecommunications, 42, 57, 84
Telephones
  electronic system, 49
  rotary systems, 49
Term, 44, 46
TERM, 50
Terminal mode, 42, 45, 46, 48, 49, 50, 52, 54, 55, 56, 57, 71
Terminator, line plug, 69
Text, 7
TEXT, 9, 11, 15, 17, 108
Text
  block, 83
  control block, 81, 83
  editor, 96
  Insertion Message, 81, 83
  mode, 11
  processor, 11, 29, 31, 38, 87
TIM, 81, 83
TIME$, 14, 172, 189, 241
TIME$ OFF, 187, 242
TIME$ ON, 187, 242
TIME$ STOP, 187, 242
Toggle, 5, 13
Token, ASCII, 116
Tone, 109
  audio, 17
  control, 30
Top menu, 73
Traffic, telecommunications, 84
Transient data, 130

Transmission
protocol, 83
speeds, 8, 47
Transmit data, 245
Trapping, error, 122
Tree diagram, 73
TYMNET, 69, 72, 76, 77, 78

UCS, 80
Unconditional branch, 99
Underlining, 31
Unidimensional array, 138
Unit, arithmetic/logic, 88
Up, 47, 51
Uploading, 36, 47, 51, 57
User Communication System, 80
USPS, 80, 83

VAL, 127, 182, 242
Values, ASCII, 98
Variable concatenation, 131
Variables, 88
alphabetic, 88, 129
double-precision, 130
integer, 130
Variables *(Contd.)*
numeric, 88, 89, 129
reversed string, 14
single-precision, 130
string, 14, 88, 130
VARPTR, 183, 187, 242
Vernier adjustment, 7
Videotex, 75
VISA, 73
Vital statistics, 4
Volatile memory, 10
VolksWriter, 9
Volume control, 30

Wait, 46, 51
Weather forecasts, 69
Widgets, 207
WIDTH, 19
Width, 31, 47
Word length, 44, 48, 49
Word processing, 9, 11
WordStar, 9
Wordwrap, 20

XOFF, 47, 49, 51, 83
XON, 47, 49, 51, 83

THE TRS-80 MODEL 100 COMPUTER

Kenniston W. Lord

Radio Shack has taken the computer off the desk and put it into the briefcase with their incredibly powerful TRS-80 Model 100 Computer. Through this guide to this computer, you'll fnd out how to take full advantage of all the features built into the machine—more than most desktop computers have. The book's three parts discuss specific features of the TRS-80 Model 100. In the first part you'll find out about the TEXT processor, the ADDReSS function, and the SCHEDuLe function. Part II examines the BASIC programming language. Part III looks at some advanced BASIC concepts—the mathematical functions and the graphics functions. With this book, you'll realize what a powerful business tool the TRS-80 Model 100 Computer really is. In fact, it's called the Micro Executive Workstation. You'll discover new and effective ways to use your new briefcase computer and wonder how you managed without it.

The table of contents includes:

Part I: Tools for the Executive
1 Features and Definitions
2 Text Processor
3 Nonprogramming Capabilities
4 On-Line Services
Part II: Tools for the Programmer
5 Aw, Programming's Not So Tough
6 Developing and Manipulating Data
Part III: Math and Graphics
7 Screen Play
8 Graphics
Appendices

Cover Design by Debbie Balboni

A Reston Computer Group Book
RESTON PUBLISHING COMPANY, INC.
A Prentice-Hall Company
Reston, Virginia

0-8359-7859-1